UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

Uma Arquitetura Híbrida com Aprendizagem para o Desenvolvimento de Agentes de Software

Adriana Leite Costa

São Luís - MA
2017

Adriana Leite Costa
Mestre em Engenharia de Eletricidade na área de Ciência da Computação
Universidade Federal do Maranhão, 2009.

Tese apresentada ao programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão como parte dos requisitos para a obtenção do título de Doutor em Engenharia Elétrica na área de Ciência da Computação.
Orientadora: Profª. Drª. Rosario Girardi.

São Luís - MA
2017

Adriana Leite Costa

Uma Arquitetura Híbrida com Aprendizagem para o Desenvolvimento de Agentes de Software

Tese aprovada em 14 de agosto de 2017

Profa. María Del Rosario Girardi Gutierrez, Dra.
Universidade Federal do Maranhão
(Orientadora)

Prof. Evandro de Barros Costa, Dr.
Universidade Federal de Alagoas
(Membro da Banca Examinadora)

Prof. Angelo Perkusich, Dr.
Universidade Federal de Campina Grande
(Membro da Banca Examinadora)

Prof. Francisco José da Silva e Silva, Dr.
Universidade Federal do Maranhão
(Membro da Banca Examinadora)

Prof. Denivaldo Cicero Pavão Lopes, Dr.
Universidade Federal do Maranhão
(Membro da Banca Examinadora)

Costa, Adriana Leite.
Uma Arquitetura Híbrida com Aprendizagem para o Desenvolvimento de Agentes de Software / Adriana Leite Costa. - 2017.
160 f.

Orientador(a): María Del Rosario Girardi Gutierrez.
Tese (Doutorado) - Programa de Pós-graduação em Engenharia de Eletricidade/ccet, Universidade Federal do Maranhão, São Luís, 2017.

1. Agentes de Software Híbridos com Aprendizagem. 2. Arquiteturas de Agentes de Software. 3. Ontologias. 4. Raciocínio Baseado em Casos. 5. Sistemas para Detecção de Intrusões. I. Gutierrez, María Del Rosario Girardi. II. Título.

A minha avó Ana Pinheiro Costa (in memorian).

AGRADECIMENTOS

A Deus que é a força maior a quem recorro em momentos de alegria e de tristeza.
À professora Dra. Rosario Girardi, por sua grande contribuição na realização deste trabalho e também pela paciência e incentivo.
Aos professores que aceitaram participar da minha banca de doutorado por suas relevantes contribuições com sugestões e críticas que me permitiram melhorar a qualidade do trabalho: Prof. Dr. Francisco Silva e Silva, Prof. Dr. Denivaldo Lopes, Prof. Dr. Evandro Costa e Prof. Dr. Angelo Perkusich.
A todos os atuais e antigos colegas do grupo de pesquisa GESEC. Em especial, a Rayane pelo trabalho em conjunto e amizade durante a fase do seu mestrado, ao Geovane e a Alana pela ajuda com a apresentação e por todo apoio de sempre.
Ao Prof. Dr. Paulo Novais por sua participação nas minhas bancas de qualificação e tese e a todos os alunos do grupo de pesquisa IsLAB, pela oportunidade do convívio com pesquisadores tão motivados e dedicados. Em especial ao Tiago, Ângelo e Fábio que sempre tentaram me auxiliar em tudo que podiam durante a minha estada na UMinho.
Ao Ariel pela ajuda com as dúvidas iniciais sobre a área de Segurança da Informação, conversas e troca de experiência sobre as publicações.
À Pedriana e ao Alex Barradas pela atenção, apoio e incentivo.
A todos os colegas da DGTI e do Núcleo de Suporte do IFMA. Em especial, ao Claudio Fernandes, Alessandra Melo e Fábio Lima que sempre se mostraram compreensivos com as exigências do doutorado me liberando para realizar as atividades quando foram necessárias.
Aos antigos colegas do Departamento de Informática, em especial ao “Seu Lima” e a Lourdinha pelo incentivo e as professoras Eveline e Evaldinólia pela carta de recomendação e incentivo.
A minha família, em especial, a minha mãe Josilene e o meu pai Edivaldo (in memorian), minha irmã Andréa por suas inúmeras contribuições com o inglês e com revisões do texto, minha avó Dalva e meus tios e primos pelos muitos momentos que tive que me afastar do convívio familiar devido as atividades do doutorado.
À Bianca, por todo apoio e incentivo durante todas as etapas deste trabalho e pelas importantes sugestões quanto à escrita e normatização do texto.
Ao meu amigo Pedro por todo apoio, pelos materiais e conversas.
Por fim, às agências de fomento CAPES e FAPEMA, que foram de grande ajuda para o custeio de algumas despesas da pesquisa.

“A simplicidade é a extrema sofisticação. ”
Leonardo da Vinci

RESUMO

Os agentes de software representam uma evolução do software tradicional, tendo a capacidade de controlar seu próprio comportamento e agir com autonomia. Tipicamente, os agentes de software agem de forma reativa, onde as percepções e ações são predefinidas no momento da sua concepção, ou de forma deliberativa, onde a ação correspondente para uma determinada percepção é encontrada em tempo de execução através de um processo de raciocínio. Os agentes deliberativos não necessitam que todo o conhecimento seja predefinido, ao contrário, a partir de um conhecimento inicial eles conseguem inferir novo conhecimento. Todavia, em muitos casos, para encontrar uma ação apropriada a uma determinada percepção eles levam muito tempo, gerando um alto custo computacional. Como solução a esse problema, apresentamos neste trabalho uma arquitetura híbrida com aprendizagem para o desenvolvimento de agentes de software híbridos. Os agentes híbridos, que combinam comportamento reativo e deliberativo, são uma opção melhor para estruturar os agentes de software. As principais vantagens da arquitetura tese é o aprendizado do comportamento reativo, mais rápido e eficiente, através de interações do agente com o seu ambiente e a sua consequente adaptabilidade ao ambiente. O agente se adapta ao ambiente na medida em que aprende novo comportamento reativo a partir de comportamento deliberativo frequente. A arquitetura tese foi avaliada através do desenvolvimento de estudos de casos no domínio da segurança da informação utilizando o raciocínio baseado em casos, ontologias para a representação do conhecimento do domínio de estudo e aprendizagem supervisionada para geração automática de regras reativas. Os resultados obtidos com os estudos de casos realizados confirmaram uma efetividade maior e um menor tempo de resposta do agente híbrido com aprendizagem em relação tanto ao comportamento isolado de um agente reativo ou deliberativo bem como de um agente híbrido sem aprendizagem no domínio da detecção de intrusões em redes de computadores. A partir da especificação e avaliação da arquitetura híbrida com aprendizagem supervisionada no domínio da Segurança da Informação, foi generalizada uma arquitetura de referência para o desenvolvimento de agentes híbridos com aprendizagem. Em trabalhos futuros, pretende-se avaliar esta arquitetura de referência em outros domínios, com outros tipos de raciocínio e técnicas de aprendizagem para avaliar o seu impacto na produtividade e qualidade do desenvolvimento de agentes de software híbridos.

Palavras-chave: Agentes de Software Híbridos com Aprendizagem, Arquiteturas de Agentes de Software, Raciocínio Baseado em Casos, Sistemas para Detecção de Intrusões, Ontologias

ABSTRACT

Software agents represent an evolution of traditional software entities, having the ability to control their own behavior and acting with autonomy. Typically, software agents act reactively, where actions and perceptions are predefined at design time, or in a deliberative way, where the corresponding action for a given perception is found at run time through reasoning. Deliberative agents do not need all knowledge to be predefined; on the contrary, from an initial knowledge they can infer new knowledge. However, to find an action appropriate to a particular perception, they take a long time, generating a high computational cost. As a solution to this problem, a hybrid architecture with learning for the development of hybrid software agents is presented in this work. Hybrid agents combine reactive and deliberative behavior taking advantage of the speed of reactive behavior and the reasoning capability of the deliberative one are a better option for structuring software agents. The main advantages of the proposed architecture are learning of the reactive behavior, faster and more efficient, through the interactions of the agent with its environment and its consequent adaptability to the environment. The agent adapts to the environment as it learns new reactive behavior from frequent deliberative behavior. The proposed architecture was evaluated through the development of case studies in the information security domain using case-based reasoning, ontologies for the representation of domain knowledge and supervised learning for automatic generation of reactive rules. Results obtained with the case studies performed confirmed a greater effectiveness and a shorter response time of the hybrid agent with learning regarding both the reactive or deliberative agent as well as a hybrid agent without learning in the intrusion detection in computer networks domain. From the specification and evaluation of the hybrid architecture with supervised learning in the Information Security domain, a reference architecture for the development of hybrid agents with learning was generalized. In future works, we intend to evaluate this reference architecture in other domains, with other types of reasoning and learning techniques to evaluate its impact on the productivity and quality of the development of hybrid software agents.

Keywords: Hybrid Learning Software Agents, Software Agent Architectures, Case-Based Reasoning, Intrusion Detection Systems, Ontologies

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ACT-R	Adaptive Control of Thought–Rational
BDI	Belief, Desire and Intentions
CBR	Case Based Reasoning
CLARION	Connectionist Learning with Adaptive Rule Induction Online
FTP	File Transfer Protocol
GAODT	Goal-Oriented Application Ontology Development Technique
GESEC	Grupo de Engenharia de Software e Engenharia do Conhecimento
HIDS	Host Intrusion Detection System
HyLAA	Hybrid and Learning Agent Architecture
HyLARA	Hybrid and Learning Agent Reference Architecture
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
InteRRaP	Integration of Reactive Behavior and Rational Planning
JESS	Java Expert System Shell
KB	Knowledge Base
KBCP	Knowledge Base Case Problem
KQML	Knowledge Query and Manipulation Language
MADAE-IDE	Multi-agent Domain and Application Engineering – Integrated Development Environment
MADAE-Pro	Multi-agent Domain and Application Engineering Process
NCP	New Case Problem
NIDS	Network Intrusion Detection System
NSL-KDD	Network Security Lab – Knowledge Discovery and Data Mining
ONTOID	Ontology for the Development of Case-based Intrusion Dectection System
OWL	Web Ontology Language
PASSI	Process for Agent Societies Specification and Implementation
POD	Ping of Death
RBC	Raciocínio Baseado em Casos
URL	Uniform Resource Locator

1. INTRODUÇÃO

O aumento da complexidade das aplicações de software tem requerido a adoção de novos paradigmas de desenvolvimento. Os agentes de software, por possuírem características como autonomia, mobilidade, sociabilidade e habilidade de aprendizado constituem uma forma adequada de abordar essa crescente complexidade.

Um agente de software é uma entidade computacional capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente através de executores [60]. Um sistema multiagente é formado por vários agentes que interagem entre si. Os agentes são capazes de simular a interação social humana, muitas vezes, realizando atividades sociais avançadas como a cooperação, coordenação e negociação [78].

Uma arquitetura de software é uma solução computacional para um problema que define como os componentes de um sistema de software interagem, proporcionando assim uma visão geral da estrutura e do comportamento do sistema. O desenvolvimento da arquitetura de um sistema multiagente tem dois produtos principais: a arquitetura da sociedade multiagente e a arquitetura particular de cada agente da sociedade. No projeto da arquitetura da sociedade são definidos os agentes que irão fazer parte da aplicação multiagente e suas interações. Nesta tarefa, a ênfase está na definição dos mecanismos de cooperação e coordenação dos agentes. No projeto do agente, procura-se definir a arquitetura interna de cada agente, de forma a satisfazer os requisitos funcionais e não funcionais da aplicação multiagente.

As arquiteturas de um agente de software são classificadas, conforme o seu tipo de comportamento [60][76][78] e a forma como são mapeadas as percepções para as ações. Podemos agrupar essas arquiteturas em três grupos principais: reativas, deliberativas e híbridas. As arquiteturas reativas são aquelas que suportam comportamento reflexivo ou instintivo e as deliberativas são aquelas que suportam diferentes formas de raciocínio. Já as arquiteturas híbridas apresentam os dois tipos de comportamento, reativo e deliberativo, em um único agente.

Normalmente, o conhecimento de um agente de software é representado em uma base de conhecimento, que inclui o conhecimento acerca do domínio, muitas vezes compartilhado com outros agentes da sociedade e o conhecimento interno, necessário para realizar as ações individuais do agente. Pela sua reusabilidade, uma das formas mais efetivas de representar as bases de conhecimento dos agentes é através de ontologias.

Uma ontologia é frequentemente definida como a especificação de uma conceitualização [22]. A conceitualização refere-se à abstração de uma parte do mundo (um domínio) onde são representados os conceitos relevantes e seus relacionamentos. A especificação é formal, permitindo uma representação e processamento semântico do conhecimento.

Os agentes de software têm três tipos de comportamentos: reativo, deliberativo e híbrido. Os agentes que tem a habilidade de agir imediatamente ao perceber uma ação e, para isso, utilizam um conjunto de regras de condição-ação são denominados reativos. Os agentes deliberativos são os que decidem em tempo de execução qual ação adequada a tomar frente a uma determinada percepção e, para isso, usam algum tipo de raciocínio. Já os agentes híbridos podem agir de forma tanto reativa quando deliberativa. O ambiente em que o agente vai estar inserido, classificados na seção 2.3 do capítulo 2, deve ser levado em consideração ao se decidir qual tipo de arquitetura interna o agente deverá ter. Por exemplo, nos ambientes estáticos, onde as percepções e ações dos agentes são previamente conhecidas, o tipo de arquitetura do agente deve ser reativa. No entanto, em ambientes dinâmicos, o agente será mais efetivo e terá um melhor desempenho se tiver a capacidade de decidir em tempo de execução quando não houver uma solução pré-estabelecida e de aprender a partir destas experiências.

O domínio de aplicação utilizado para avaliação da arquitetura híbrida com aprendizagem é o da segurança da informação, particularmente, o relativo à detecção de intrusões em redes de computadores. Este domínio foi escolhido por ser dinâmico, no qual muitas ameaças se repetem com frequência e novas ameaças surgem, tornando, com isso, a capacidade de aprendizagem relevante. A segurança da informação [9] visa proteger a informação de vários tipos de ameaças e seus princípios básicos são a disponibilidade, confidencialidade, autenticidade e integridade. Isto é, o seu objetivo é garantir que as informações estejam sempre disponíveis aos usuários, que sejam confiáveis, verdadeiras e que apenas pessoas autorizadas tenham acesso ao seu conteúdo. O conhecimento acerca do domínio da segurança da informação, dos sistemas de detecção de intrusão e do raciocínio baseado em casos foi representado na ontologia de aplicação ONTOID (“Application Ontology for the Development of Case-based Intrusion Detection Systems”) [43][44].

Uma arquitetura de referência [13] especifica uma solução arquitetural genérica para o desenvolvimento de arquiteturas de software específicas que normalmente inclui componentes comuns a um conjunto de arquiteturas de software, um vocabulário compartilhado, uma metodologia de mapeamento e boas práticas de projeto para o desenvolvimento de arquiteturas

específicas também conhecidas por realizações. Como parte deste trabalho, uma arquitetura de referência foi generalizada a partir do desenvolvimento de agentes híbridos com aprendizagem.

1.1 Motivação

Segundo Wooldridge [78], a maioria dos pesquisadores da área de Inteligência Artificial acredita que nenhuma abordagem puramente deliberativa nem puramente reativa seria o melhor caminho para a construção de agentes com capacidade de ações autônomas. Assim, naturalmente, tem aumentado o volume de estudos no sentido de definir arquiteturas híbridas que combinem características das arquiteturas deliberativas com as reativas.

Russel e Norvig [60] fazem a seguinte pergunta no Capítulo 27 "Qual das arquiteturas definidas no Capítulo 2 deve um agente usar?", sendo que as arquiteturas definidas no Capítulo 2 são as arquiteturas reativas ou deliberativas, tais como o agente reativo simples, o agente reativo com estado, o agente com objetivos, etc. Os autores respondem a esse questionamento com "todas elas!", indicando as arquiteturas híbridas, que unem vários tipos de arquiteturas tradicionais como as arquiteturas ideais. Entretanto, estes autores não foram além dessa definição na análise, compreensão e especificação das arquiteturas de agentes de software híbridas.

Além de combinar componentes deliberativos e reativos, alguns trabalhos recentes sugerem também características avançadas tais como o aprendizado, modelagem de emoções e diversas formas semânticas de representação do conhecimento que simulam o funcionamento da memória humana.

A ideia inicial da arquitetura híbrida com aprendizagem tese neste trabalho deriva do conceito de arquitetura híbrida ilustrado na Figura 1, no qual um agente inicialmente deliberativo vai se tornando mais reativo através da sua interação com o ambiente, isto é, o agente aprende comportamento reativo ao longo do tempo. Desta forma, ao aprender um novo comportamento reativo previamente realizado de forma deliberativa, ele se adapta as mudanças do ambiente evitando a necessidade de atualização manual da base de conhecimento com novas regras reativas.

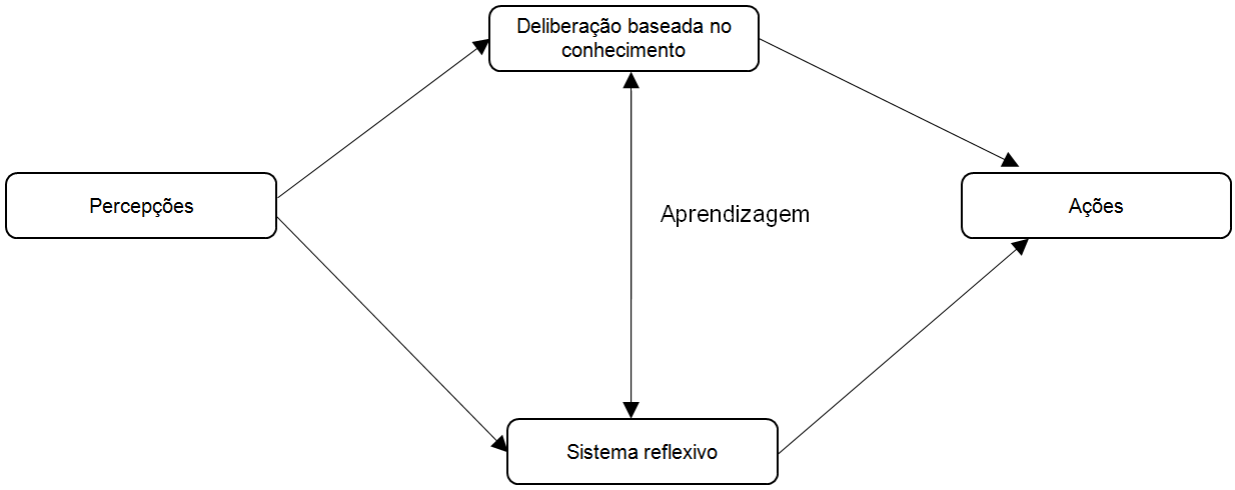


Figura 1. Arquitetura híbrida genérica. Fonte: Adaptado de Russel e Norvig [60]

Já existem na literatura alguns trabalhos propondo arquiteturas híbridas [5][34][35][54][67]. O diferencial da arquitetura tese neste trabalho é a aprendizagem de comportamento reativo a partir de comportamento deliberativo frequente. Assim, ao longo do tempo, ações que são frequentemente realizadas de forma deliberativa, vão sendo aprendidas e passam a serem executadas de forma reativa e, portanto, mais rapidamente. Além disso, os trabalhos do estado da arte não contemplam o ciclo de vida de desenvolvimento completo de um agente de software. A ausência de fases do ciclo de vida de desenvolvimento tais como a Engenharia de Requisitos e o Projeto Arquitetural compromete a qualidade do agente desenvolvido. Por outro lado, também já existem processos para o desenvolvimento de agentes de software que suportam várias fases do ciclo de vida [12][15][73][38][51], mas que não consideram o desenvolvimento de agentes de software híbridos.

Por outro lado, a especificação de uma arquitetura de referência para o desenvolvimento de agentes de software híbridos também é necessária porque as arquiteturas híbridas atuais têm um baixo nível de abstração, isto é, elas foram estruturadas para serem utilizadas com determinadas técnicas de aprendizagem e implementações muito específicas. Por sua vez, as arquiteturas de referência, são reusáveis, possibilitando o desenvolvimento de agentes com diferentes habilidades e inseridos em diferentes domínios de aplicação. Nesse sentido, uma motivação secundária deste trabalho de tese é contribuir para o avanço no desenvolvimento de agentes de software através da definição de uma arquitetura híbrida de referência que suporte o aprendizado de comportamento reativo a partir do deliberativo e habilidade de se adaptar às mudanças do ambiente através da aprendizagem.

1.2 Objetivos

1.2.1 Objetivo geral

Contribuir para o avanço no estado da arte no desenvolvimento de agentes de software em ambientes dinâmicos através da especificação e avaliação de uma arquitetura para o desenvolvimento de agentes de software híbridos com aprendizagem que seja mais efetiva que as arquiteturas reativas e deliberativas.

1.2.2 Objetivos específicos

- Análise dos avanços e desafios ainda existentes no desenvolvimento de arquiteturas de agentes de software;
- Análise do estado da arte das áreas de conhecimento envolvidas na construção de um agente de software, como raciocínio automático, aprendizagem de máquina e representação do conhecimento;
- Desenvolvimento de uma arquitetura híbrida que, através do aprendizado, apresente comportamento flexível e adaptável à evolução do ambiente;
- Avaliação da arquitetura desenvolvida;
- Generalização da arquitetura híbrida específica em uma arquitetura de referência.

1.3 Hipótese de pesquisa

A aprendizagem de comportamento reativo a partir de comportamento deliberativo frequente torna os agentes híbridos mais efetivos, especialmente em ambientes dinâmicos, do que as simples arquiteturas de agente reativas e deliberativas quando executadas independentemente ou que as arquiteturas híbridas sem aprendizagem.

1.4 Metodologia da pesquisa

A metodologia utilizada para realização desta tese está sintetizada em quatro etapas, descritas a seguir:

Etapa 1: Definição do tema e sua delimitação

Durante esta etapa, o tema a ser investigado “arquiteturas híbridas” foi escolhido baseado, em sua relevância para o estado da arte e para o grupo de pesquisa ao qual a aluna faz parte. Em seguida, o tema foi delimitado de acordo com a literatura na área de agentes de

software e engenharia de software multiagente. Paralelamente, uma pesquisa inicial quanto aos trabalhos relacionados foi realizada. Nesta etapa, também foram definidos objetivos e hipótese de pesquisa do trabalho.

Etapa 2: Fundamentação teórica e trabalhos relacionados

Após a etapa de definição do tema, um estudo mais profundo dos conceitos essenciais relacionados ao desenvolvimento de agentes de software, principalmente na definição de arquiteturas no nível detalhado foi realizado. Nesta etapa foram estudados tópicos como Engenharia de Software Orientada a Agentes, Engenharia de Conhecimento e suas contribuições na construção de sistemas multiagente, técnicas de aprendizagem de máquina e suas aplicações no desenvolvimento de agentes de software. Nesta etapa, também foi analisado de forma mais minuciosa os trabalhos relacionados, isto é, outras arquiteturas híbridas para o desenvolvimento de agentes de software.

Etapa 3: Especificação de uma solução ao problema abordado

Nesta etapa, foi especificada uma solução ao problema identificado na Etapa 1, isto é, uma arquitetura híbrida particular com aprendizagem foi definida. Neste momento, também foi escolhido o domínio de aplicação, a detecção de intrusões em redes de computadores baseado nas características da arquitetura. A arquitetura foi aplicada então a este domínio e um agente de software foi desenvolvido para detecção de intrusões utilizando raciocínio baseado em casos e uma ontologia representando casos de intrusões foi instanciada e utilizada como base de conhecimento pelo agente. Após a aplicação da arquitetura híbrida no domínio da detecção de intrusões em redes de computadores, uma arquitetura de referência foi generalizada, isto é, uma arquitetura independente de domínio.

Etapa 4: Avaliação experimental da solução proposta

A última fase consistiu em avaliar a arquitetura híbrida desenvolvida na Etapa 3 para avaliar se a mesma demonstrava a hipótese de pesquisa. Isso foi feito através da definição e realização de quatro estudos de casos no domínio da segurança da informação que serviram para avaliar comparativamente a efetividade e o desempenho de um agente reativo ou deliberativo quando executado independentemente, de um agente híbrido sem aprendizagem e de um agente

híbrido com aprendizagem desenvolvido utilizando a arquitetura HyLAA. Os resultados obtidos com os estudos de casos demonstraram a hipótese de pesquisa.

1.5 Organização do manuscrito

O Capítulo 2 apresenta a fundamentação teórica deste trabalho. Descreve a classificação das arquiteturas básicas de um agente de software, incluindo os componentes internos de um agente, o mecanismo de raciocínio, a representação do seu conhecimento e os mecanismos de comunicação. Neste capítulo, também é apresentada uma análise comparativa das principais arquiteturas híbridas teses na literatura. O Capítulo 3 aborda a parte central deste manuscrito que é o desenvolvimento de uma arquitetura híbrida com aprendizagem denominada HyLAA ("Hybrid and Learning Agent Architecture") e a sua aplicação no desenvolvimento de um agente de software híbrido para a detecção de intrusões em uma rede de computadores. O Capítulo 4 apresenta a avaliação da arquitetura HyLAA, incluindo a metodologia utilizada, os resultados obtidos e uma discussão sobre esses resultados. O Capítulo 5 apresenta a generalização da arquitetura HyLAA em uma arquitetura de referência denominada HyLARA ("Hybrid and Learning Agent Reference Architecture"), para o desenvolvimento de agentes de software híbridos com aprendizagem. Por fim, o Capítulo 6, traz as considerações finais acerca dos resultados obtidos no trabalho, suas principais contribuições, limitações e os trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

A autonomia, a capacidade de aprendizado, a mobilidade e a sociabilidade são as habilidades comumente encontradas em um agente de software, sendo a autonomia aquela que os distingue das aplicações tradicionais.

Os agentes necessitam interagir para realizar algumas tarefas que estão além das suas capacidades individuais, formando uma sociedade de agentes, onde cada agente ou grupo de agente é responsável por determinados tipos de ações. Segundo Jenninngs et al. [29], o termo "sistemas multiagente" refere-se a todos os tipos de sistemas compostos de vários componentes com certo grau de autonomia e tem como característica o fato de nenhum agente ter controle total sobre a sociedade pois, mesmo que algumas sociedades de agentes possuam estruturas hierárquicas, onde alguns agentes gerenciam os demais, não há um controle central; cada agente tem um ponto de vista limitado e os dados são descentralizados, isto é, nenhum agente é detentor de todo o conhecimento, mas especialista em determinadas tarefas.

Um agente com arquitetura genérica, ilustrado na Figura 2 percebe o ambiente através de seus sensores, interpreta essa percepção, transformando-a em uma sentença. Em seguida, ele realiza a unificação entre a sentença-percepção com as sentenças da base de conhecimento. Após encontrar a sentença que representa a ação, essa sentença será interpretada e executada no ambiente. O mapeamento da percepção para a ação, representado na Figura 2 pelo ponto de interrogação, poderá ser realizado de forma direta no caso do agente ser do tipo reativo ou através de um processo de raciocínio se ele for do tipo deliberativo.

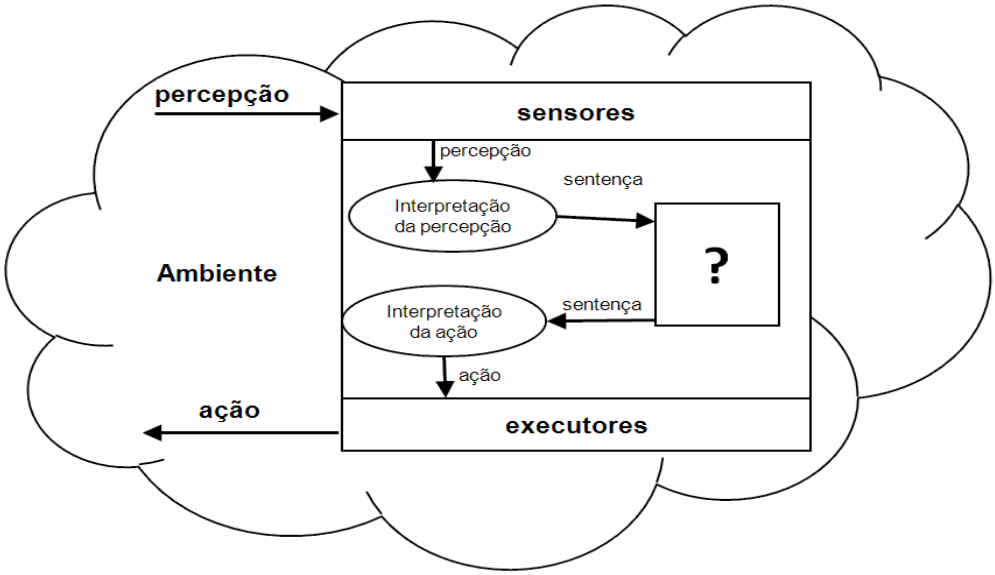


Figura 2. Agente com arquitetura genérica que interage com seu ambiente através de sensores e atuadores. Fonte: Adaptado de Russel e Norvig [60]

2.1 Arquiteturas de agentes de software

Durante o projeto de um agente de software é estabelecido quais são seus componentes internos e como esses componentes se relacionam entre si. Existem diversas classificações na literatura para essas arquiteturas de acordo com quais componentes são utilizados. Nas próximas subseções são apresentadas as classificações de arquiteturas de agentes de software mais conhecidas.

2.1.1 Classificação de Russel e Norvig

Russel e Norvig definem quatro tipos básicos de arquitetura de software [60]:

- Agentes reflexivos simples;
- Agentes reflexivos simples baseados em modelos;
- Agentes baseados em objetivos;
- Agentes baseados na utilidade.

O agente do tipo reflexivo simples é considerado o tipo de agente mais simples dentre aqueles definidos por Russel e Norvig. Nesse tipo de arquitetura acontece um mapeamento direto de uma percepção para uma ação, sendo que a ação do agente é realizada somente em função da sua percepção atual, pois o mesmo não guarda o estado do ambiente. A Figura 3 ilustra um agente reflexivo simples composto por sensores, atuadores e regras de mapeamento de uma condição para uma ação.

A Figura 4 ilustra um algoritmo genérico descrevendo o funcionamento básico deste tipo de agente. Quando o sensor do agente tem uma nova percepção, a mesma é interpretada e é feita sua unificação com as regras de <condição, ação> da base de conhecimento, definidas no projeto do agente, sendo que a ação da regra selecionada é depois interpretada para que possa ser utilizada pelo atuador na execução de uma determinada ação no ambiente.

Na Figura 5, apresenta-se um exemplo de aplicação da arquitetura genérica de um agente reflexivo simples em um agente denominado aspirador de pó. Esse agente tem como ambiente duas salas denominadas A e B que podem ter o estado “suja” ou “limpa”. Quando o agente perceber que a sala A está suja ele deverá realizar a ação “limpar” e, em seguida, se dirigir à sala B e ao perceber que a mesma também está “suja” ele irá executar a ação “limpar”. O funcionamento do agente aspirador de pó é descrito no algoritmo da Figura 6.

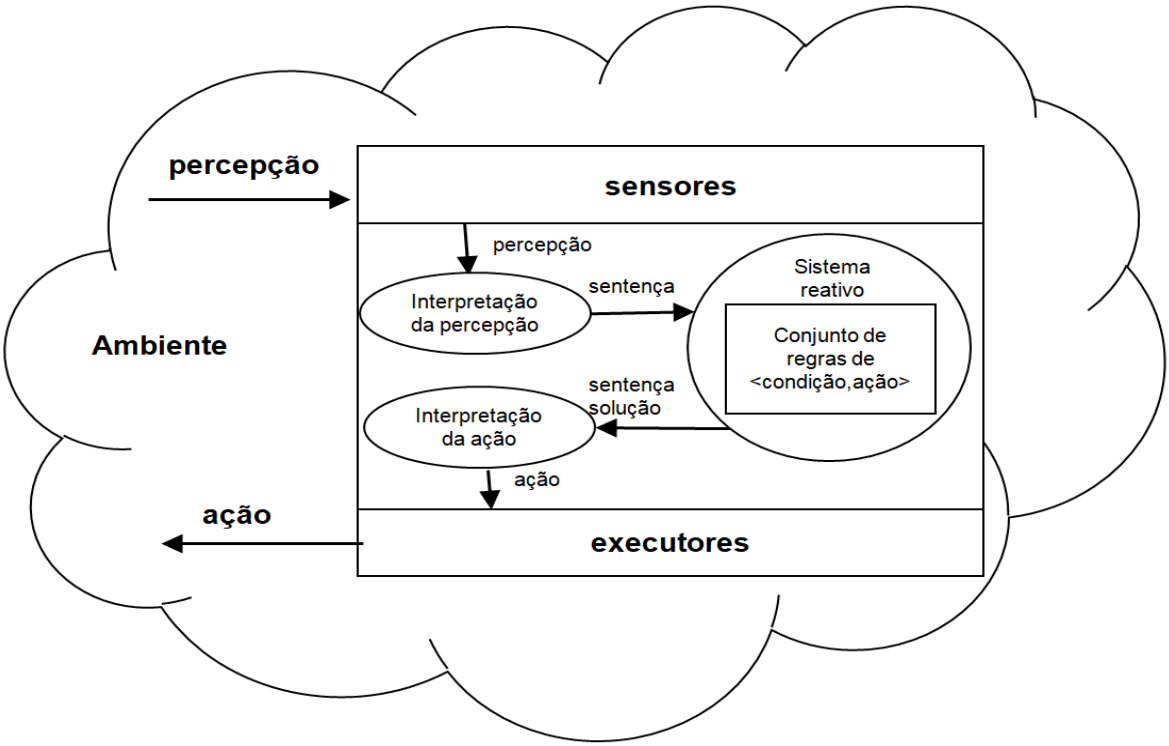


Figura 3. Arquitetura de um agente reflexivo simples Fonte: Adaptado de Russel e Norvig [60]

```
function Skeleton-Agent(percept)

static: memory, the agent's memory of the world
memory ← Update-memory(memory, percept)
action ← Choose-Best_action(memory)
memory ← Update-memory(memory, action)

return action
```

Figura 4. Algoritmo genérico de um agente reflexivo simples. Fonte: Adaptado de Russel e Norvig [60]

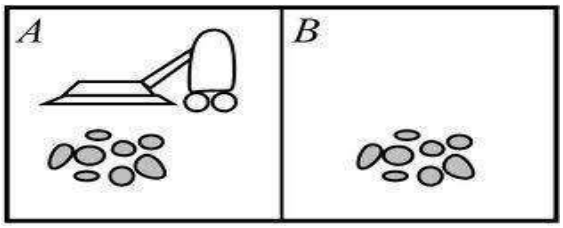


Figura 5. Exemplo de agente: mundo do aspirador de pó. Fonte: Adaptado de Russel e Norvig [60]

```
function REFLEX-VACUUM-AGENT([location,status])

if status = Dirt then return Suck
else if location = A then return Right
else if location = B then return Left

returns an action
```

Figura 6. Algoritmo do agente aspirador de pó. Fonte: Adaptado de Russel e Norvig [60]

O agente do tipo reflexivo simples baseado em modelos, anteriormente denominado de agente com estado, difere do agente reflexivo simples por manter o estado do ambiente. Assim, não só a percepção atual é levada em consideração como também as percepções e ações anteriores. Esse tipo de agente atualiza constantemente o seu estado a partir das novas percepções do ambiente e suas ações são executadas de acordo com essa atualização.

Na Figura 7 está representada a arquitetura de um agente baseado em modelo e na Figura 8 temos um exemplo de um algoritmo descrevendo o funcionamento desse tipo de agente. Basicamente, ele percebe o ambiente, atualiza o seu estado interno, procura a ação adequada à percepção atual, executa a ação e depois atualiza novamente o seu estado com a ação executada.

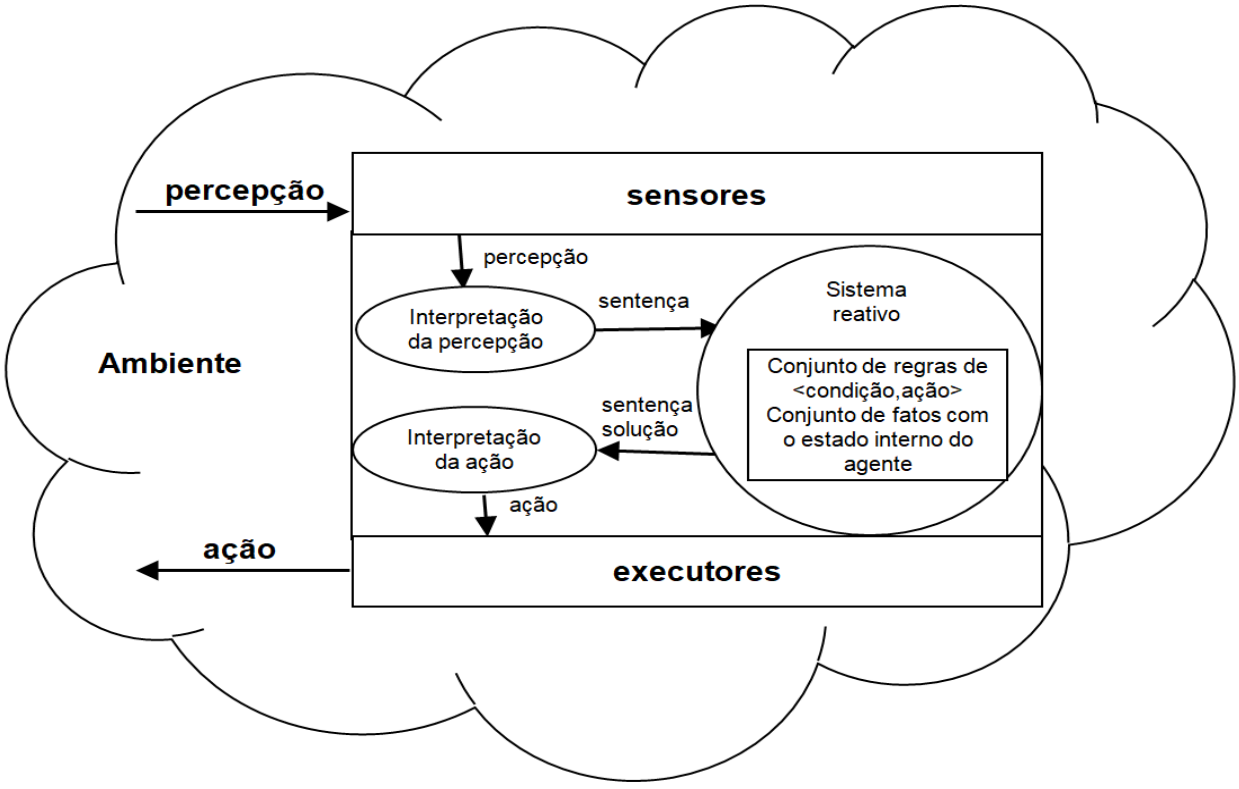


Figura 7. Arquitetura de um agente reativo baseado em modelos. Fonte: Adaptado de Russel e Norvig [60]

```
function Reflex-Agent-With-State(percept) returns   action

static: state, a description of the current world state
        rules, a set of condition-action rules
state ← Update-State(state, percept)
rule ← Rule-Match(state, rules)
action ← Rule-Action (rule)
state ← Update-State(state, action)

return action
```

Figura 8. Algoritmo genérico de um agente baseado em modelos. Fonte: Adaptado de Russel e Norvig [60]

O agente do tipo baseado em objetivos (Figura 9) além de manter o histórico dos estados do ambiente como o agente reflexivo baseado em modelos, também tem um objetivo a ser atingido. Para atingir um objetivo o agente pode ter que executar uma ação ou uma sequência de várias ações. Esse tipo de agente possui um mecanismo de raciocínio, diferentemente dos agentes reativos que possuem apenas regras simples de condição-ação. No entanto, por essa característica, os agentes do tipo baseado em objetivos, normalmente, apresentam menor desempenho frente aos agentes reflexivos devido ao maior tempo de processamento requerido para raciocinar em comparação ao tempo necessário para a execução de uma regra de condição-ação. Um algoritmo descrevendo o funcionamento do agente baseado em objetivos é ilustrado na Figura 10.

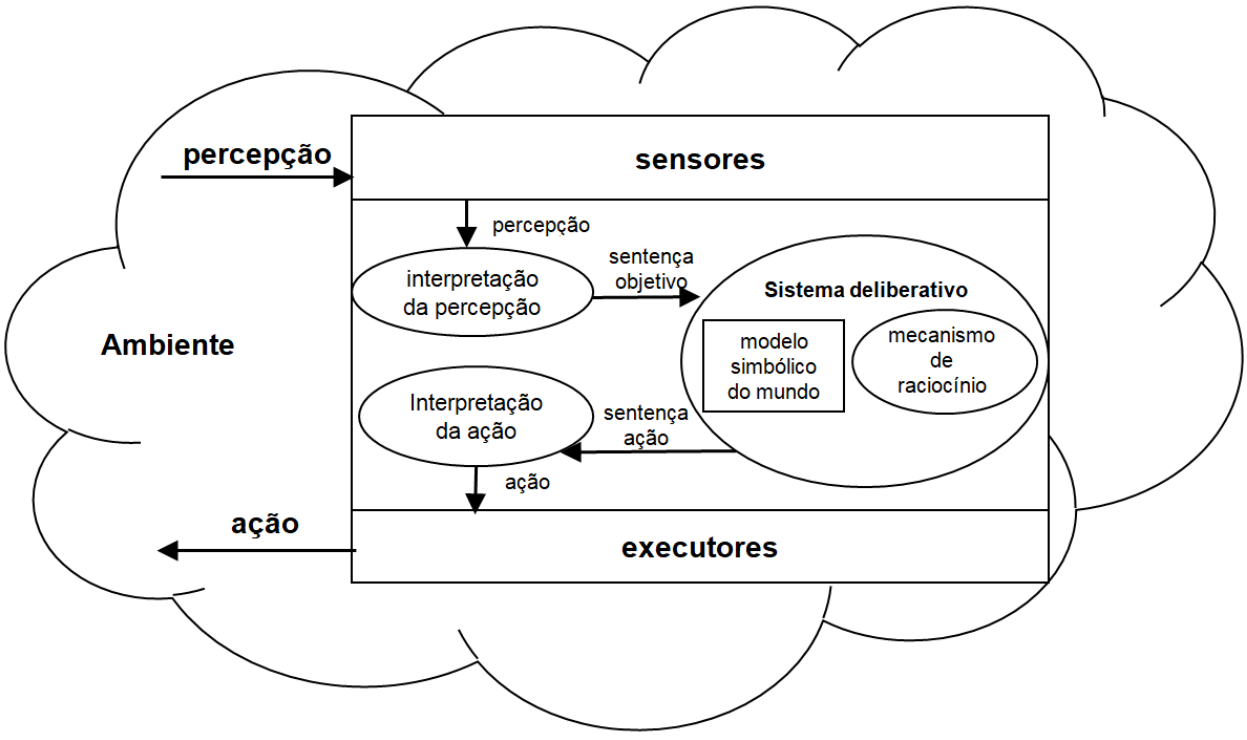


Figura 9. Arquitetura de um agente baseado em objetivos. Fonte: Adaptado de Adaptado de Russel e Norvig [60]

```
function  Goal-Based-Agent (percept) returns action
static: state, a description of the current world state

state ← Update_State(state, percept)
goal ← Define_Goal(state)
action ← Get_Action_From_Plan(goal)
state ← Update_State(state, action)

return action
```

Figura 10. Algoritmo genérico de um agente baseado em objetivos. Fonte: Adaptado de Russel e Norvig [60]

O agente do tipo baseado na utilidade (Figura 11) além de manter o estado do ambiente (como os agentes reflexivos baseados em modelos) e possuir um objetivo a ser atingindo (como os agentes baseados em objetivos), também comporta uma medida de utilidade que é utilizada para mensurar o nível de sucesso com o qual o objetivo foi atingido. Por exemplo, um agente cujo objetivo é ir de uma "cidade A" até uma "cidade B" pode atingir esse objetivo levando duas ou quatro horas para concluir o percurso. Através da definição de uma medida de utilidade de tempo, o agente saberá que a realização do percurso em menos tempo é melhor. Um algoritmo com o funcionamento básico de um agente baseado na utilidade é descrito na Figura 12.

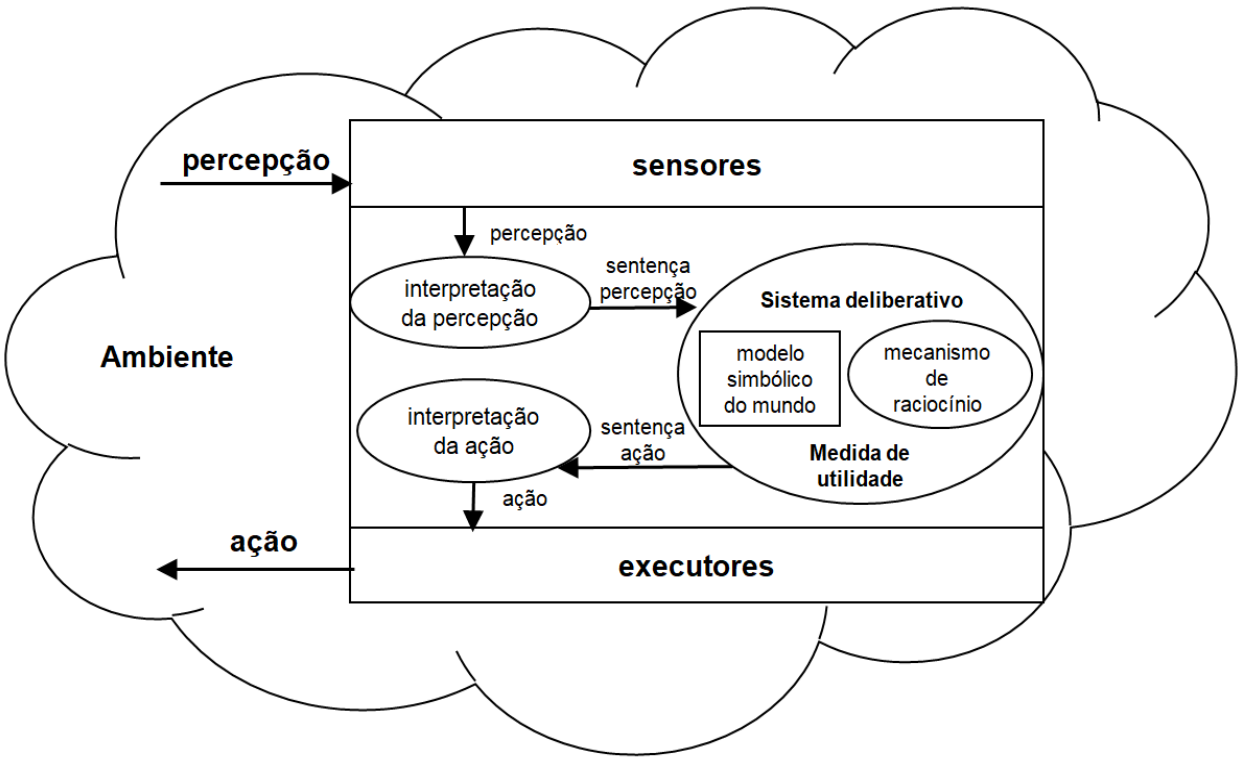


Figura 11. Arquitetura de um agente baseado na utilidade. Fonte: Adaptado de Adaptado de Russel e Norvig [60]

```
function  Utility-Based-Agent (percept) returns action
static: KB, goal, search-space

state ← Update_State(state, percept)
 goal ← Define_Goal(state, utility_measure)
 action ← Get_Action_From_Plan(goal)
 state ← Update_State(state, action)

return action
```

Figura 12. Algoritmo genérico de um agente baseado na utilidade. Fonte: Adaptado de Russel e Norvig [60]

2.1.2 Classificação de Wooldridge

Wooldridge [78] propõe uma classificação de arquiteturas onde os agentes são separados em quatro categorias principais:

- Agente com raciocínio dedutivo;
- Agente com raciocínio prático;
- Agente reativo;
- Agente híbrido.

Agente com raciocínio dedutivo

A arquitetura de um agente com raciocínio dedutivo é aquela em que o agente possui um modelo simbólico do mundo e o seu comportamento é representado explicitamente, normalmente utilizando lógica. O agente manipula a sua base de conhecimento através da dedução. Neste tipo de arquitetura uma das principais desvantagens é que a dedução lógica pode requerer um tempo considerável para ser realizada. Com isso, o ambiente do agente pode se modificar durante o processo de raciocínio e aquela ação não ser mais adequada.

Agente com raciocínio prático

O agente com raciocínio prático baseia-se na ideia de que o agente não age somente a partir de raciocínio lógico, mas também de acordo com suas crenças, desejos e intenções de forma análoga aos seres humanos. Um exemplo de um raciocínio puramente lógico é: “Todos os homens são mortais”, “Sócrates é um homem”, “logo, Sócrates é mortal”. No caso de um agente com raciocínio prático, quando existe um conjunto de ações dada uma percepção, ocorre um processo de decisão baseado nas suas crenças, desejos e intenções. As crenças de um agente são seu conhecimento do mundo; seus desejos fornecem algum tipo de estado desejável a ser atingido; e suas intenções são as ações que ele mesmo decidiu realizar para alcançar os seus

desejos. Esse tipo de agente também é conhecido por agente BDI (“Belief, Desire and Intentions”).

Para entendimento do agente com raciocínio prático, em [76] o seguinte exemplo simples é dado: um aluno quando deixa a universidade com uma primeira graduação tem uma decisão a tomar sobre o que fazer com sua vida. O processo de decisão se inicia buscando entender e selecionar uma das opções disponíveis. Por exemplo, se esse aluno tem um bom histórico escolar, então uma opção é tornar-se acadêmico. Outra opção é trabalhar em uma empresa. Após a geração do grupo de alternativas, deve-se optar por uma delas, e se comprometer com ela, de modo que esta escolha vem a ser sua intenção. As intenções alimentam o raciocínio prático futuro do agente. Por exemplo, se ele decidiu que quer se tornar um acadêmico, então deve se comprometer com esse objetivo e dedicar tempo e esforço para alcançá-lo.

Na Figura 13 estão ilustrados os principais elementos de uma arquitetura com raciocínio prático. Essa arquitetura é composta por um conjunto de crenças atuais que representam as informações que o agente tem no momento sobre seu ambiente; por uma função de revisão de crenças, que a partir da entrada de uma percepção e das crenças atuais do agente determina um novo conjunto de crenças atualizado; por uma função de geração de opções , que determina as opções de ações disponíveis para o agente, com base nas suas crenças atuais sobre o ambiente e suas intenções atuais; um conjunto de opções atuais, representando as ações disponíveis para o agente; uma função de filtro que representa o processo de deliberação do agente e que determina as intenções do agente com base em suas crenças atuais, desejos e intenções; um conjunto de intenções atuais, representando o objetivo atual do agente e, finalmente, uma função de seleção de ação que determina uma ação a ser executada com base nas intenções atuais.

Como principais vantagens das arquiteturas BDI [78], estão a sua proximidade conceitual com o processo de decisão humano, bem como o entendimento informal das noções de crença, desejo e intenção.

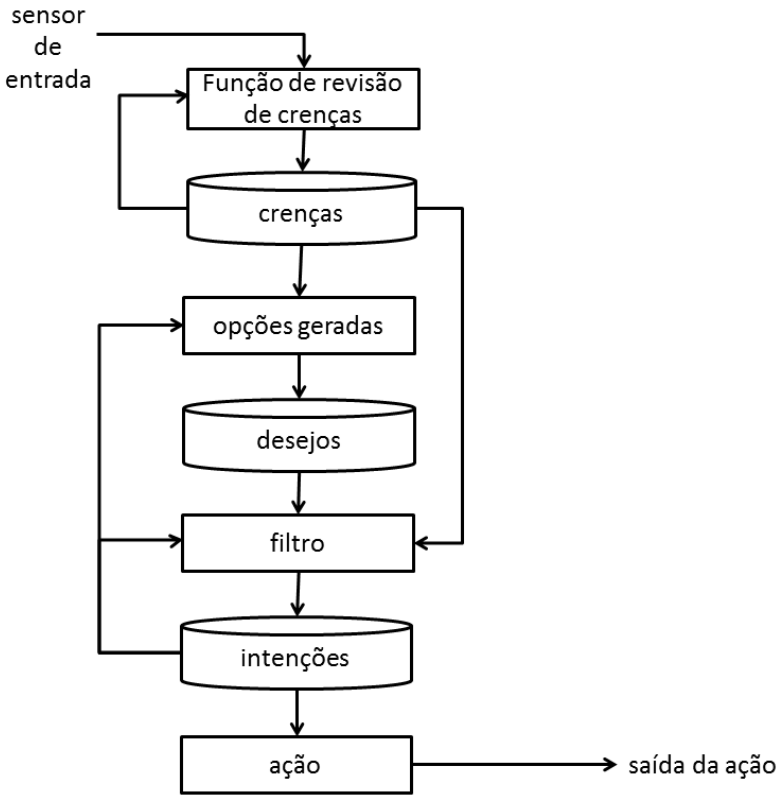


Figura 13. Arquitetura genérica de um agente BDI. Fonte: Adaptado de Wooldridge [78]

Agentes reativos

A arquitetura de agente reativo foi definida com o objetivo de suprir a principal lacuna das arquiteturas baseadas em lógica: o tempo de processamento. O objetivo da arquitetura reativa é que o agente consiga apresentar comportamento inteligente através de um conjunto de comportamentos rápidos e simples. O conhecimento do agente e do seu comportamento não é necessariamente representado em lógica e o agente não realiza nenhum tipo de raciocínio. No entanto, neste tipo de arquitetura, o conhecimento sobre as regras do domínio devem ser conhecidas a priori. Na arquitetura reativa, um conjunto de regras de mapeamento direto da situação (percepção) para ação é definido por um especialista no domínio. Algumas arquiteturas reativas são organizadas em camadas com níveis de abstração distintos, onde as camadas mais baixas possuem um nível de prioridade maior, isto é, ações críticas são realizadas por essas camadas. As camadas da arquitetura também podem ser independentes, isto é, cada uma pode processar uma percepção e realizar uma ação. Nesse caso, as ações podem ser executadas no ambiente em paralelo.

A Figura 14 ilustra uma arquitetura reativa em camadas, onde cada camada possui um sensor e uma saída, podendo assim atuar individualmente. Uma desvantagem da utilização desse

tipo de arquitetura é que a adição de várias camadas reativas aumenta a complexidade e torna difícil o gerenciamento das interações entre essas camadas.

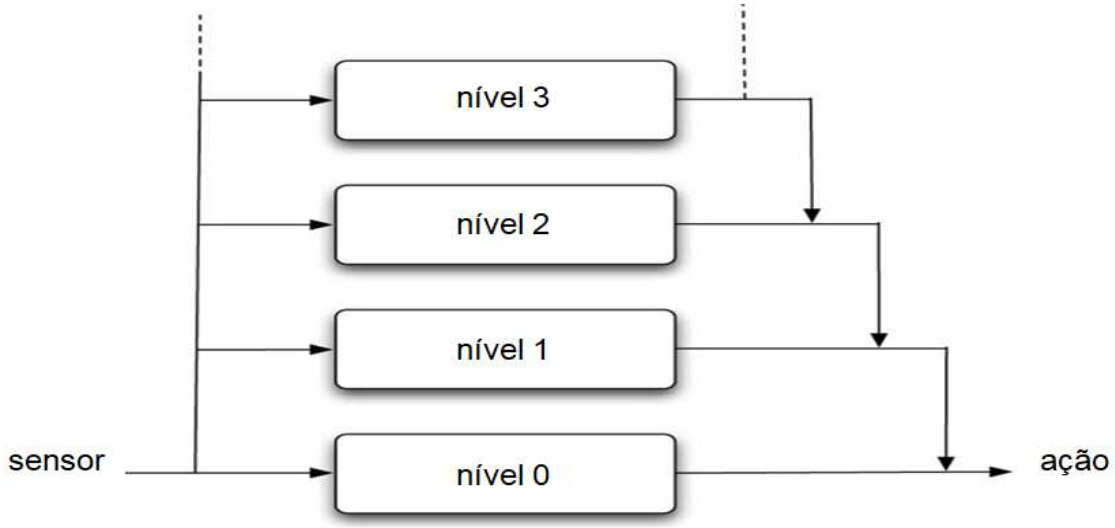


Figura 14. Arquitetura de um agente do tipo reativo. Fonte: Wooldridge [78]

Agentes híbridos

As arquiteturas de agente híbrido surgiram da necessidade de reunir em um único agente o comportamento reativo e deliberativo. Wooldridge classificou essas arquiteturas em dois grupos: as arquiteturas híbridas em camadas horizontais e as arquiteturas híbridas em camadas verticais, ilustradas na Figura 15.

Nas arquiteturas híbridas em camadas horizontais (Figura 15a), cada camada de software está ligada diretamente ao sensor e a saída de ação. Nesse tipo de arquitetura cada camada funciona como um agente. Uma vantagem das arquiteturas organizadas horizontalmente em camadas é a separação clara dos tipos de comportamentos dos agentes dentro da arquitetura, onde cada camada pode atuar de forma independente das demais, inclusive em paralelo. Um dos problemas desse tipo de arquitetura é que o comportamento geral do agente pode não ser coerente, portanto para resolver isso, geralmente uma camada de controle também é desenvolvida para assegurar um comportamento global coerente.

Nas arquiteturas híbridas em camadas verticais as percepções e as ações são tratadas por mais de uma camada. As camadas verticais ainda podem ser subdivididas em: arquiteturas de uma passagem (Figura 15.b) e arquiteturas de duas passagens (Figura 15c).

Na arquitetura em camadas verticais de uma passagem, o fluxo de controle passa sequencialmente através de cada camada até chegar à camada final onde é gerada a saída da ação. Na arquitetura em camadas verticais de duas passagens, a informação flui até chegar à última camada (primeira passagem) e o controle então flui de volta para baixo (segunda passagem).

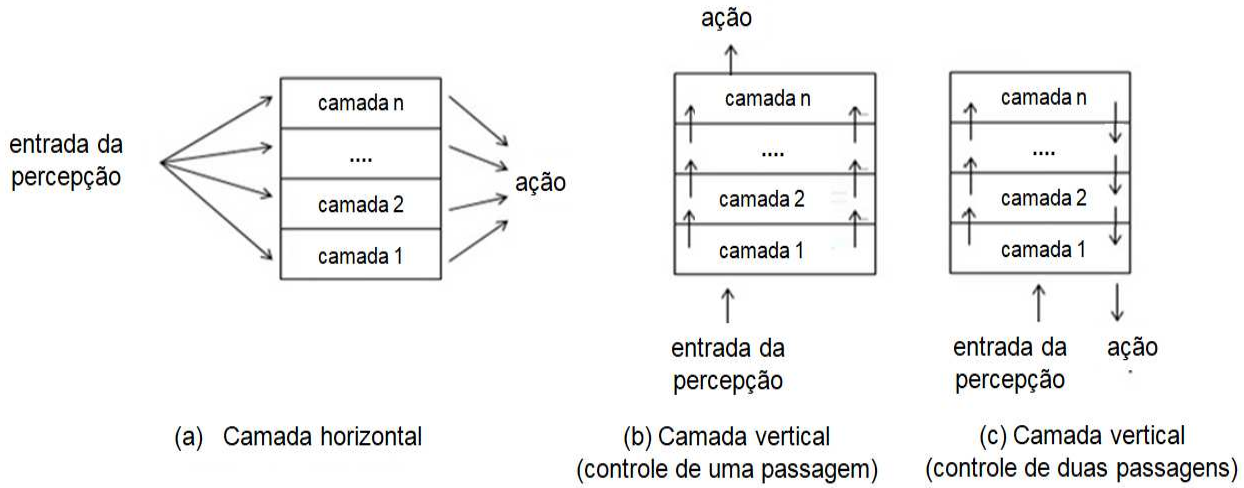


Figura 15. Arquitetura em camadas horizontais e verticais. Fonte: Adaptado de Wooldridge [78]

2.1.3 Classificação de Kendall

Kendall [31] definiu dois tipos de arquiteturas de agentes organizados em camadas: agentes em camadas deliberativos e agentes reativos. Uma diferença entre as definições desse autor e as dos demais é que ele representa os agentes através de padrões de software.

A Figura 16 representa o agente reativo definido por Kendall. Ele é dividido em três componentes fixos e três opcionais. Os componentes fixos são aqueles que todos os agentes deverão possuir obrigatoriamente, como o componente responsável por perceber o ambiente ("camada sensores"), o conjunto de regras reativas ("camada regras") e o módulo responsável pela execução das ações no ambiente ("camada ação"). Quando existir apenas um agente, esses três componentes são suficientes. No entanto, quando se tratar de uma aplicação multiagente, um ou mais dos componentes opcionais poderão ser usados. Kendall define o seguinte padrão para a estruturação do agente reativo (Figura 16):

Problema: Como pode um agente reagir a um estímulo do ambiente ou a uma requisição de outro agente quando não há representação simbólica e nenhuma solução conhecida?

Forças: Um agente tem que ser capaz de responder a um estimulo ou um pedido, sendo que pode não haver uma representação simbólica da aplicação.

Solução: Um agente reativo não possui modelos simbólicos internos do seu ambiente; ele age usando um tipo de comportamento estímulo/resposta e reunindo informações sensoriais, mas suas camadas de crenças e de raciocínio são reduzidas a um conjunto de regras de condição-ação.

Usos conhecidos: agentes reativos têm sido amplamente utilizados para simular o comportamento das sociedades de formigas.

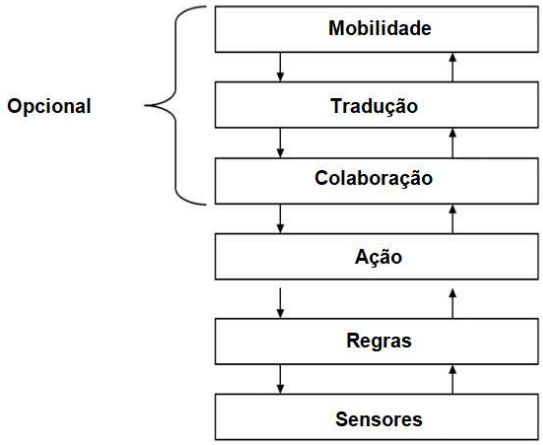


Figura 16. Arquitetura de um agente reativo proposto por Kendall. Fonte: Adaptado de Kendall [31]

Kendall [31] especifica o seguinte padrão de agente em camadas deliberativo (Figura 17):

Problema: Como pode o comportamento do agente ser melhor organizado e estruturado em software? Qual arquitetura de software suporta melhor o comportamento dos agentes?

Forças: Um sistema agente é complexo e abrange vários níveis de abstração; há dependências entre os níveis de vizinhos, com dois sentidos de fluxo de informação; a arquitetura de software deve abranger todos os aspectos da agência; a arquitetura deve ser capaz de lidar com qualquer comportamento simples e sofisticado.

Solução: Os agentes devem ser decompostos em camadas porque o comportamento de maior nível ou mais sofisticado depende de capacidades de nível mais baixo, porque as camadas só dependem de seus vizinhos e porque há dois fluxos de informações entre camadas vizinhas.

Na camada sensorial o agente percebe seu ambiente através de sensores. As crenças dos agentes estão baseadas em estímulos sensoriais, assim, na camada de crenças, as percepções são

mapeadas para sentenças lógicas que são incluídas na base de conhecimento do agente. Na camada de raciocínio, quando confrontado com um problema, o agente raciocina sobre a base de conhecimento para determinar o que fazer, selecionando uma ação específica para executar no ambiente. Quando o agente decide sobre uma ação, ela pode ser realizada diretamente, ou pode requerer a colaboração de outros agentes. Uma vez que a abordagem de colaboração é necessária, uma mensagem é enviada para outro agente (utilizando a camada de mobilidade).

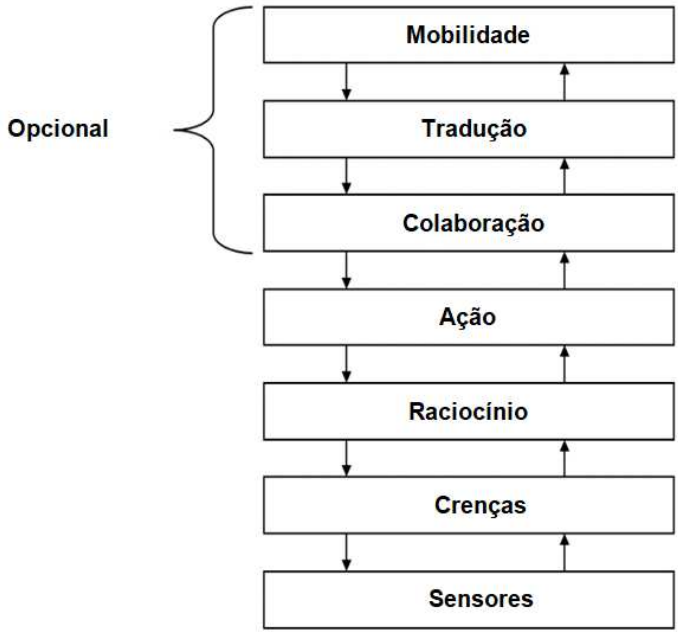


Figura 17. Agente em camadas de Kendall. Fonte: Adaptado de Kendall [31]

2.1.4 Agentes com aprendizagem

Nesta seção, o agente com aprendizado, um tipo mais avançado de agente é introduzido, juntamente com as técnicas utilizadas para a construção do seu componente de aprendizado. Em seguida, são apresentadas as arquiteturas híbridas e um comparativo entre essas arquiteturas.

A ideia por trás de aprendizagem é que as percepções devem ser usadas não só para agir, mas também para melhorar a capacidade do agente para agir no futuro [60]. Os agentes de software tradicionais não possuem aprendizagem, isto é, eles apenas agem de acordo com as percepções pré-definidas em seu projeto. Quando surgem percepções novas, o agente deve ser reprogramado.

No entanto, o agente com aprendizado (Figura 18), possui um componente de aprendizagem e através dele é possível que o agente, em tempo de execução, mude seu comportamento de acordo com as mudanças no ambiente.

Segundo Russel e Norvig [60], um agente que possui a capacidade de aprendizagem possui quatro componentes básicos:

1. Elemento de desempenho;
2. Crítico;
3. Elemento de aprendizagem;
4. Gerador de problemas.

O componente "elemento de desempenho" corresponde a um agente genérico, ele percebe e age no ambiente e pode ser tanto reativo quanto deliberativo. O elemento de aprendizagem é responsável pela realização de melhorias no comportamento do agente. O componente crítico tem como função informar ao elemento de aprendizagem sobre o êxito do agente de acordo com um padrão fixo de desempenho. O elemento de aprendizagem usa o "realimentação" do crítico para determinar como o elemento de desempenho deve ser modificado para ter um comportamento com melhor desempenho no futuro. O componente gerador de problemas é responsável por sugerir ações exploratórias que conduzam a novas experiências ao agente.

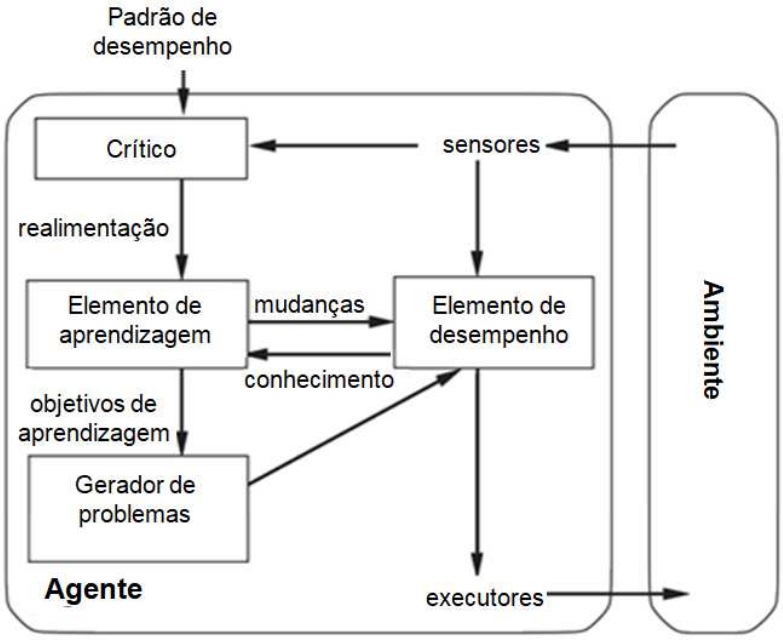


Figura 18. Arquitetura de um agente com aprendizado. Fonte: Adaptado de Russel e Norvig [60]

O agente táxi automatizado é um exemplo de agente com aprendizado [60]. O elemento de desempenho do agente táxi automatizado consiste em conhecimento e procedimentos que formam a base de conhecimento do agente e é utilizada para selecionar suas ações de acordo com as percepções do ambiente. O componente "crítico" desse agente observa o ambiente, avalia o resultado da ação no ambiente no ambiente ("realimentação") e envia este resultado ao elemento de aprendizagem. Por exemplo, após o táxi automatizado bater no carro da frente durante um dia de chuva, o crítico percebe informações do ambiente sobre o resultado dessa ação que pode ser, por exemplo, uma multa. Então, o crítico avalia essas informações e avisa ao elemento de aprendizagem que essa não foi uma boa ação. A partir dessa experiência, se ela se repetir com certa frequência o "elemento de aprendizagem" deve ser capaz de formular uma regra contendo a informação de que essa ação não foi boa e o elemento desempenho é modificado através da inclusão da nova regra. O "gerador de problema" pode sugerir novas experiências, como experimentar os freios em diferentes intensidades de acordo com as condições climáticas e com a velocidade do veículo.

2.2 Correspondência entre as terminologias

Na Tabela 1 está definida a correspondência entre os tipos de agentes definidos por Russel e Norvig [60], Wooldridge [78] e Kendall [31]. O agente reflexivo simples e o agente com estado definidos por Russel e Norvig correspondem ao agente reativo de Wooldridge e ao agente reativo de Kendall. A diferença principal é que Wooldridge e Kendall definem que as arquiteturas podem ser dispostas em camadas com aumento do nível de abstração.

Os agentes baseados em metas e baseado na utilidade definidos por Russel e Norvig correspondem ao agente com raciocínio dedutivo de Wooldridge e ao agente em camadas de Kendall. Russell e Norvig definem uma medida de desempenho e utilidade enquanto Kendall organiza sua arquitetura deliberativa em camadas. Em geral, essas arquiteturas são chamadas arquiteturas deliberativas e usam algum tipo de raciocínio. Wooldridge e Kendall definem uma arquitetura com raciocínio prático, também conhecido como arquitetura BDI, cujos conceitos principais não são abordados por Kendall e Russell e Norvig. Wooldridge define um agente híbrido, também conhecido como arquitetura em camadas, já Russel e Norvig apresentam apenas algumas ideias gerais sobre os agentes híbridos.

Tabela 1. Correspondência entre a classificação de Russel e Norvig e de Wooldridge

Russel e Norvig [60]	Wooldridge[78]	Kendall[31]
Agente reflexivo simples	Agente reativo	Agente reativo
Agente reflexivo baseado em modelos		
Agente baseados em objetivos e Agente baseado na utilidade	Agente com raciocínio dedutivo	Agente em camadas
-	Agente com raciocínio prático	
Arquitetura híbrida	Agente híbrido	

A seguir são apresentados os tipos de ambientes em que os agentes estão inseridos e os componentes internos dos tipos básicos de agente reativo e deliberativo.

2.3 Tipos de ambientes

Um agente de software está inserido em um ambiente e a correta caracterização do ambiente em que ele irá atuar é muito relevante durante o seu projeto. Por exemplo, um ambiente estável não requer habilidades como o aprendizado, mas em um ambiente dinâmico esta habilidade pode ser imprescindível para o bom desempenho do agente.

Classificar o ambiente do agente também é importante para definir o tipo de arquitetura do agente e as técnicas de implementação que poderão ser utilizadas. Assim, nesta seção será apresentada uma classificação para os ambientes dos agentes.

Segundo Russel e Norvig [60] antes mesmo do projeto dos componentes internos do agente é recomendado que sejam definidas as principais características do ambiente onde o agente irá atuar. Os autores definem um conceito mais abrangente chamado de ambiente de tarefa, composto de medida de desempenho, ambiente, atuadores e sensores. Por exemplo, no agente táxi automatizado, a medida de desempenho poderia ser minimizar o consumo de combustível, reduzir o tempo gasto na viagem e não violar as leis de trânsito. O ambiente seria composto pelas estradas, os outros veículos, os pedestres, os clientes do táxi, etc. Os atuadores seriam a direção, acelerador, freio, sinal, buzina, etc. Já os sensores poderiam ser o velocímetro, câmeras, GPS, etc. Apesar de o exemplo dado ser de um agente situado em um ambiente real, o ambiente pode ser artificial como a Internet. Nesse caso, os sensores, poderiam ser entradas pelo teclado e os atuadores mensagens apresentadas numa tela de computador.

Os ambientes de tarefa se classificam como a seguir [60]:

Completamente observável ou parcialmente observável: um ambiente é completamente observável para um agente, se seus sensores têm acesso ao estado completo do ambiente em cada instante. Neste ambiente o agente percebe todos os aspectos que são relevantes para a escolha da ação, sendo que um aspecto é relevante se ele influencia na medida de desempenho do agente. Em ambientes completamente observáveis o agente não precisa manter um estado interno para controlar o ambiente, pois a todo instante pode observá-lo completamente. Já os ambientes parcialmente observáveis são aqueles em que os sensores dos agentes não conseguem ter acesso completo ao ambiente devido a alguma limitação. Por exemplo, um agente aspirador de pó encarregado de limpar uma casa não consegue saber a qualquer instante se todos os cômodos da casa estão limpos devido à existência de paredes. Já um agente encarregado de limpar uma única sala pode ter acesso ao estado completo do ambiente. No primeiro caso, o agente necessita armazenar no seu estado interno quais cômodos já foram limpos, enquanto que no segundo caso o agente poderia simplesmente observar se o ambiente está sujo ou limpo.

Determinístico ou estocástico: "Se o próximo estado do meio ambiente estiver completamente determinado pelo estado atual e a ação executada pelo agente, então dizemos que o ambiente é determinista; de outra forma, é estocástico." [60]. Por exemplo, considerando que o estado anterior do agente aspirador de pó é "chão da sala 1 sujo" e ele executa a ação "limpar sala 1", o resultado será "chão da sala 1 limpo". No entanto, se o ambiente for estocástico existe a possibilidade de interferência externa, então mesmo após limpar a sala 1 não há garantias que o estado da mesma continue igual, pode ser, por exemplo, que no mesmo instante uma pessoa jogue um papel no chão, causando um estado inesperado para o agente. Existem alguns tipos de ambiente em que há garantia da não existência de interferência interna, por exemplo, no caso dos jogos individuais, como o jogo de palavras-chave. Logo esses ambientes serão determinísticos.

Episódico ou sequencial: um ambiente é episódico quando a interação do agente com ambiente consiste apenas de uma percepção associada a uma ação (episódio). Nesse tipo de ambiente, cada episódio é interpretado independentemente dos episódios anteriores ou futuros. Por exemplo, um agente de detecção de peças defeituosas em uma linha de produção situado em um ambiente episódico, tem uma percepção "peça com defeito" e executa a ação associada "descartar peça", esse episódio não irá influenciar as demais percepções e ações dele. Já no ambiente sequencial, uma ação atual pode influenciar todas as ações futuras, assim um agente

jogador de xadrez que tem como ação determinado movimento no jogo, irá influenciar diretamente as próximas ações.

Estático ou dinâmico: "A distinção discreta / contínua pode ser aplicada ao estado do ambiente, a maneira como o tempo é tratado, quanto as percepções e ações do agente. Por exemplo, um ambiente de estado discreto, como um jogo de xadrez, tem um número finito de estados distintos" [60]. Assim, um ambiente é estático quando ele não se altera enquanto o agente está deliberando. Um jogo de palavras cruzadas é classificado como ambiente estático, pois as peças do jogo não se alteram enquanto o agente está deliberando. Já no ambiente dinâmico, acontece o contrário, ou seja, o ambiente pode se alterar enquanto o agente está deliberando. O agente táxi automatizado está situado em um ambiente dinâmico, pois enquanto ele está deliberando para escolher qual caminho seguir estão acontecendo mudanças no ambiente, por exemplo, o semáforo pode mudar de cor ou surgir um congestionamento no caminho que ele escolheu seguir. Assim, nos ambientes dinâmicos é necessário que o agente esteja constantemente monitorando as mudanças que ocorrem no ambiente para que possa se adequar a elas.

Discreto ou contínuo: um ambiente pode ser classificado em discreto ou contínuo em relação ao seu estado, ao conjunto de percepções e ações dos agentes ou a passagem do tempo. Ele será classificado como discreto em relação a uma dessas variáveis se ela possuir valores finitos ou enumeráveis, porém será considerado contínuo em relação a uma dessas variáveis quando elas puderem assumir qualquer valor dentro de um intervalo. Por exemplo, o estado de um ambiente do agente aspirador de pó é discreto, pois podemos enumerar todos os seus possíveis estados (posição X – chão limpo, posição Y – chão sujo, etc). Já o ambiente de um agente táxi automatizado tem o estado contínuo, pois, por exemplo, a localização e a velocidade do veículo podem assumir um conjunto de valores (localização X, velocidade 60, 70, 80, etc).

Agente único ou multiagente: quando existe apenas um agente no ambiente esse é classificado como ambiente de agente único, já quando existe mais de um agente no ambiente, este é classificado como multiagente. Um exemplo de um ambiente de um único agente é o do jogo de palavras-chave, já em um ambiente multiagente podemos citar o jogo de xadrez (dois agentes) e do táxi automatizado (considerando que existem outros veículos no ambiente). Essa classificação é importante, pois existem aspectos particulares aos ambientes multiagente. Por

exemplo, deverá ser definido como os agentes irão se comunicar, como eles vão estar organizados, quais suas funções, o vocabulário que vão utilizar na comunicação, etc.

O ambiente do agente pode ter várias classificações apresentadas anteriormente. Por exemplo, um agente pode estar em um ambiente classificado ao mesmo tempo como completamente observável, determinístico, episódico, estático, discreto e que possua um único agente ou em um ambiente classificado como parcialmente observável, estocástico, sequencial, dinâmico, contínuo e multiagente.

2.4 Base de conhecimento

A base de conhecimento de um agente de software ou de uma sociedade de agentes pode ser representada utilizando diferentes formalismos. A seguir são apresentadas as principais formas de representação de uma base de conhecimento comumente utilizadas por agentes de software como os frames e as ontologias.

Frames

Os frames são uma forma de representação do conhecimento introduzida por Minsky [45]. Nesse trabalho, ele apresenta os frames como uma forma de representação natural do conhecimento surgida a partir da observação de como um ser humano representa e recupera informações na sua memória. Essa ideia tem se confirmado, pois os frames, ou sistemas de frames (conjunto de frames organizados hierarquicamente) são aplicados em áreas como a lingüística onde se faz necessário que o conhecimento esteja o mais próximo possível da forma de representação humana.

Os frames são considerados por muitos autores como uma evolução das redes semânticas. Na prática, um frame representa um objeto, com suas propriedades e relacionamentos com outros objetos. Uma propriedade de um frame é denominada de slot. Por exemplo, se tivermos um frame “Pessoa”, ele provavelmente teria os slots “nome”, “idade”, “altura” e “peso”. Quando o frame for instanciado, esses slots terão um determinado valor que, por sua vez, será de um tipo especifico, por exemplo, o slot “nome” poderia ter como valor “Maria” e seria do tipo String. Um slot também pode possuir facetas. Uma faceta permite anexar informações adicionais aos slots, por exemplo, um slot idade poderia conter como faceta uma faixa de valores de 0 a 130, assim o slot idade só poderia receber valores dentro dessa faixa.

Os relacionamentos entre os frames podem ser definidos de acordo com o domínio como, por exemplo, “matriculado_em” e “casado_com” ou podem ser pré-definidos como é_um

(“*is_a”)* que representa a relação entre uma classe e sua instância (instanciação) e tipo_de (“*kind_of”)* que representa uma relação entre uma classe e sua subclasse (herança). As Figuras 19 e 20 representam, respectivamente, um conjunto de frames e sua correspondente instanciação:

```
FRAME Pessoa KIND-OF THING
 Slots   Nome TIPO String
         Idade TIPO Integer
 FRAME Aluno KIND-OF Pessoa
 Slots Matriculado_em DOMINIO Curso
 FRAME Curso KIND-OF THING
 Slots Nome TIPO String
       Codigo TIPO Integer
```

Figura 19. Exemplo de um frame representando uma pessoa

```
FRAME Curso_1 IS-A Curso
Nome VALOR “Ciência da Computação”
Codigo VALOR 234566

 FRAME Aluno_1 IS-A Aluno
 Slots
         Nome VALOR “Maria”
         Idade VALOR Pessoa_1
         Matriculado_em VALOR Curso_1
```

Figura 20. Exemplo de instância de um frame

Ontologias

As ontologias são uma maneira de se conceitualizar de forma explícita e formal os conceitos e restrições relacionados a um domínio de interesse [23]. Uma ontologia pode ser classificada de acordo com o seu nível de generalidade, como ontologia de alto-nível, domínio, tarefa ou aplicação [22]. Ontologias de alto-nível descrevem conceitos genéricos que são independentes de um domínio particular, tal como tempo. Ontologias de domínio representam conceitos explícitos, relacionados a um determinado domínio de conhecimento e os relacionamentos existentes entre eles, por exemplo, os conceitos do domínio de segurança da informação tratados neste trabalho.

As ontologias de tarefa descrevem atividades que podem ser usadas em um ou vários domínios como, por exemplo, detectar ataques à segurança da informação. Por último, as ontologias de aplicação descrevem aplicações que especializam tanto os conceitos de ontologias de domínio quanto de tarefas. Por exemplo, uma ontologia para IDS é uma aplicação que depende dos conceitos de segurança da informação (domínio) e executa as atividades de detectar e alertar (tarefas) sobre intrusões para administradores de rede. A Figura 21 representa essa classificação.

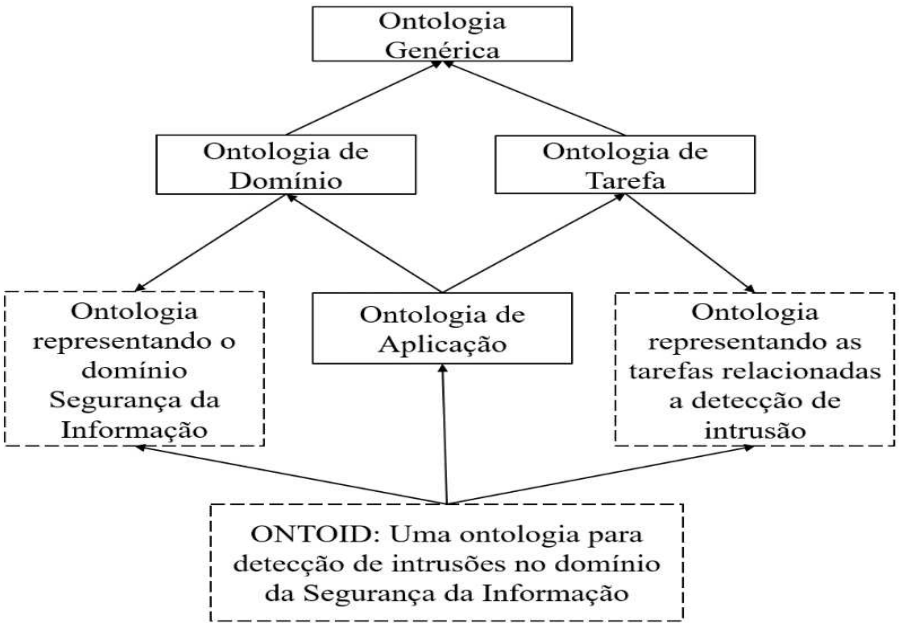


Figura 21. Classificação das Ontologias. Fonte: Adaptado de [22]

A OWL [48] é uma das linguagens para representação de ontologias mais recentes, ela é adotada pelo W3C (“World Wide Consortium”) como linguagem padrão de ontologias para Web Semântica. A OWL é baseada na lógica de descrição que por sua vez é considerada uma evolução dos frames e das redes semânticas.

As ontologias expressas em OWL podem ser de três tipos: OWL-Lite, OWL-DL e OWL-Full. O que diferencia esses três tipos de OWL (denominadas sublinguagens) é a expressividade, sendo que a OWL-Lite é a menos expressiva, a OWL-Full é a mais expressiva e a OWL-DL possui um nível intermediário de expressividade. A OWL-Full é a linguagem OWL completa e é utilizada nos casos onde é necessária uma grande expressividade. A OWL-DL é menos expressiva do que a OWL-Full e é baseada na lógica descritiva que por sua vez é baseada na lógica de primeira ordem. A OWL-Lite é uma linguagem mais simples e é utilizada nos casos onde é necessário apenas a definição de restrições em uma hierarquia de classes.

Uma ontologia em OWL tem três elementos fundamentais que são as classes, individuais (instâncias) e as propriedades (slots). Um exemplo de uma ontologia OWL construída no Protégé [28] [50], um editor de ontologias, é ilustrada na Figura 22, onde a classe “Pizza” está selecionada. Na aba “Properties” são definidos os atributos das classes e na aba “Individuals” estão as instâncias da classe. O desenvolvimento de uma ontologia no Protégé é bastante facilitado por ele possuir uma interface gráfica amigável e bastante intuitiva.

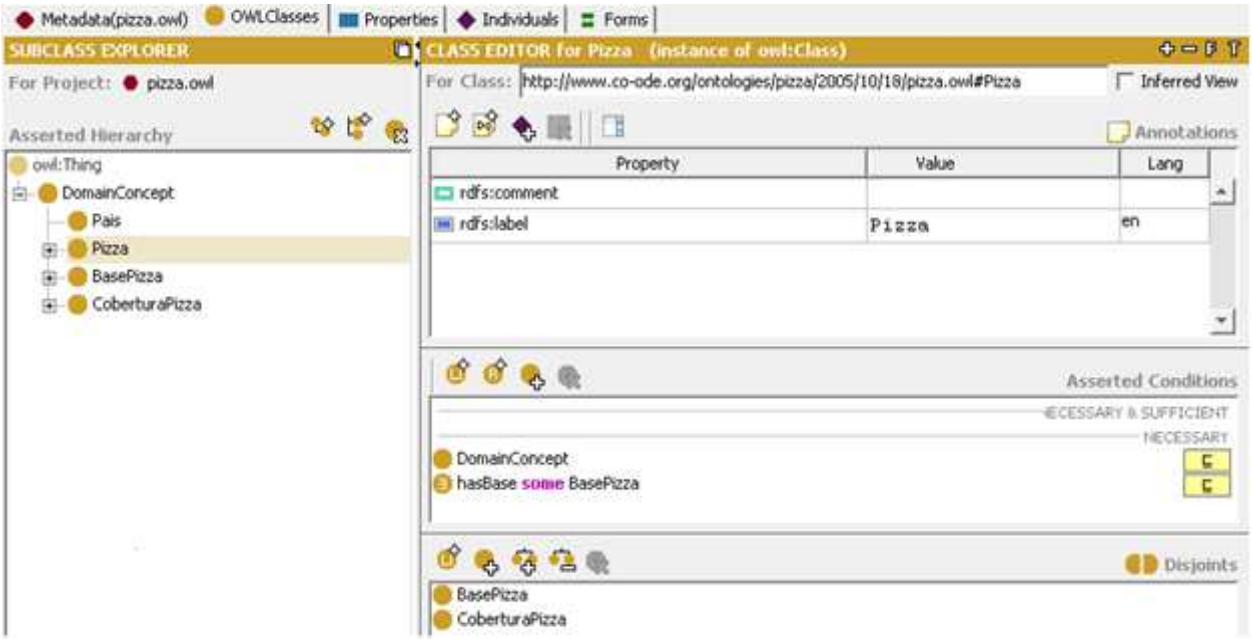


Figura 22. Exemplo de ontologia em OWL

2.5 Mecanismos de raciocínio

Raciocínio é o processo de realizar inferências sobre um conjunto de premissas a fim de obter conclusões. Existem quatro tipos principais de raciocínio: dedução, indução, abdução e analogia. Os agentes de software possuem mecanismos de raciocínio e o utilizam para realizarem ações mais complexas do que as ações reativas ou reflexivas. Nas seções seguintes são apresentadas as principais características dos quatro tipos de raciocínio mais comuns.

Dedução

A dedução é o tipo mais rigoroso de raciocínio, pois quando as premissas são verdadeiras, obrigatoriamente, a conclusão também o será. Ou seja, é impossível que sendo as premissas verdadeiras a conclusão seja falsa. Por exemplo:

> Todos os pássaros voam (Premissa 1).
> O Canário é um pássaro (Premissa 2).
> Logo, o Canário voa (conclusão).

Como no exemplo anterior, o tipo de dedução mais popular é o que parte do geral para o específico. No entanto, existem casos em que a dedução parte do específico para o específico e também do geral para o geral. Um exemplo do raciocínio deliberativo partindo do geral para o geral é o seguinte:

> Todo brasileiro é mortal (Premissa 1).
> Todo maranhense é brasileiro (Premissa 2).

Logo, todo maranhense é mortal (Conclusão).

Um exemplo do raciocínio deliberativo partindo do particular para o particular pode ser visto aplicando o *Modus ponens*:

Se P, então Q. (Se a premissa P é verdadeira então Q também é).
P. (A premissa P é verdadeira)
Portanto Q. (Portanto a premissa Q é verdadeira)

Exemplificando:
Se Maria nasceu em São Luís, então ela é Maranhense.
Maria nasceu em São Luís.
Portanto, ela é Maranhense.

Outro exemplo de dedução onde se pode partir do específico para o específico pode ser visto utilizando o Modus tollens:

Se P, então Q.(Se a premissa P é verdadeira então a premissa Q também é).
Q é falso. (A premissa Q é falsa)
Logo, P é falso. (Portanto a premissa P é falsa).

Exemplificando:

Se João nasceu em Teresina, então ele é Piauiense.
João não é Piauiense.
Portanto, ele não nasceu em Teresina.

A forma mais comum de raciocínio utilizada pelos agentes do tipo deliberativo é a dedução. Linguagens lógicas populares como Prolog [33][66] possuem mecanismos de inferência que utilizam a dedução para raciocinar sobre bases de conhecimento.

Indução

O raciocínio indutivo parte da observação de objetos e da semelhança entre suas propriedades. A partir da observação das características comuns de um conjunto limitado de objetos generaliza-se para toda uma categoria. Por exemplo:

O Canário voa (Observação 1).
O Bem-te-vi voa (Observação 2).
O Beija-flor voa (Observação n).
Logo, todos os pássaros voam. (Conclusão).

Como foi possível observar no exemplo anterior, na indução usualmente parte-se do particular para o geral. Esse processo é chamado de generalização. Também existem casos onde se parte de uma observação particular baseada no passado para uma conclusão particular. Por exemplo:

> Todos os canários que eu vi até hoje são amarelos (Observação).
> O próximo canário que eu verei será amarelo (Conclusão).

Outra característica é que nem sempre o resultado será verdadeiro, as premissas apenas sugerem uma conclusão possível. A indução é baseada na probabilidade da conclusão ser verdadeira e essa probabilidade pode variar conforme a amostragem das observações.

O raciocínio indutivo é comumente utilizado por agentes de aprendizado. As técnicas de aprendizado supervisionado e não-supervisionado, detalhadas no capítulo 4, utilizam o raciocínio indutivo.

Abdução

A abdução [71] elabora hipóteses explicativas para as observações e estas hipóteses são avaliadas. Por exemplo, a partir da observação "A rua está molhada" é possível, considerando outras observações prévias, elaborar as hipóteses de que "Choveu" ou "Um caminhão pipa passou derramando água". Então se procura validar, a partir de outras observações, uma das hipóteses a partir de experiências anteriores. Por exemplo, a partir da nova observação de que "O telhado da casa está molhado", é possível descartar a hipótese do caminhão, pois é conhecido que o caminhão pipa não seria capaz de molhar o telhado, validando assim a primeira hipótese e encontrando nela a melhor explicação.

O raciocínio abdutivo também é conhecido como inferência pela melhor explicação. Na abdução, assim como na indução, a conclusão não é uma verdade universal, mas uma probabilidade de ser verdade.

O raciocínio abdutivo é um tipo de raciocínio utilizado frequentemente por criminalistas, detetives e para diagnóstico de doenças.

Analogia

Há ainda outro processo de raciocínio denominado analogia [40], no qual parte-se do particular para o particular. Este processo ocorre com a percepção de características similares e a

classificação destas de acordo com um propósito. Por exemplo, um conjunto de indivíduos A, B e C, que possuem o seguinte conjunto de características em comum: cor branca, magros, cor do cabelo loura e olhos azuis. Considerando que o indivíduo A possui ainda como característica a estatura alta, é possível afirmar que B e C provavelmente terão esta característica mesmo que não esteja explícita na definição.

A analogia é um tipo de raciocínio muito utilizado no dia a dia das pessoas e já foi aplicada com sucesso em várias áreas do conhecimento como, por exemplo, nos primeiros esboços para construção de um avião feito por Leonardo da Vinci que imitava os mecanismos utilizados pelos pássaros para voar.

Tipos de analogia

Há duas vertentes principais de analogia: a analogia derivacional e a analogia transacional. Na analogia derivacional "linhas de raciocínio de um determinado problema são transferidas e adaptadas a um novo problema" [75]. Na analogia transacional, quando os problemas são similares, a solução de um é adaptada e reusada em outro problema.

O Raciocínio Baseado em Casos (RBC) é um tipo de analogia transacional e é uma das abordagens mais populares para o desenvolvimento de sistemas baseado no conhecimento por ser uma forma de resolução de problemas semelhante a humana [1][58]. Nesse tipo de raciocínio o conhecimento é representado através de casos, onde cada caso possui um problema e uma solução.

Para ilustrar essa definição, a Figura 23 mostra um exemplo simplificado de um caso contendo dados de um caso de enfermidade de um paciente, incluindo seus sintomas, diagnóstico e o tratamento recomendado pelo médico. Assim, os sintomas e os dados do paciente compõem o problema do caso, enquanto que o diagnóstico e o tratamento compõem a sua solução.

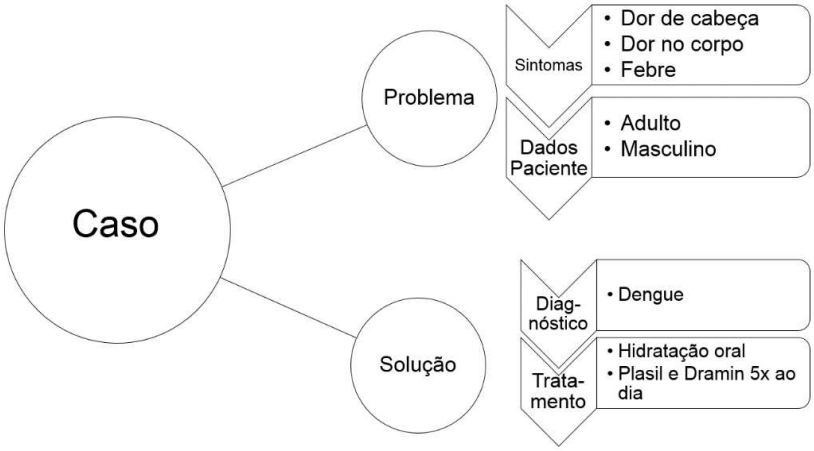


Figura 23. Exemplo de um caso de enfermidade de um determinado paciente

De acordo com [1], os principais componentes de um sistema RBC são:

- Um mecanismo baseado no conhecimento para representação de casos;
- Um mecanismo para recuperar casos e uma medida de similaridade para identificar os casos relevantes para um problema atual;
- Um mecanismo de adaptação permitindo adaptar um caso recuperado para a situação atual;
- Um mecanismo de aprendizagem permitindo que o sistema reutilize casos solucionados com sucesso para solucionar novos problemas.

A Figura 24 mostra os principais componentes de um sistema RBC. A entrada do sistema é uma consulta que consiste de um novo caso-problema descrevendo o problema a ser resolvido. Em seguida, o mecanismo de representação de casos constrói uma representação interna do novo caso, que é comparado com casos na base de casos do sistema RBC para identificar casos similares. Assim, uma solução para o novo caso-problema é obtida a partir dos casos recuperados e essa solução será eventualmente adaptada. Se esta solução for adequada para o problema, o mecanismo de aprendizagem irá avaliar a possibilidade de incluir o novo caso resolvido na base de casos.

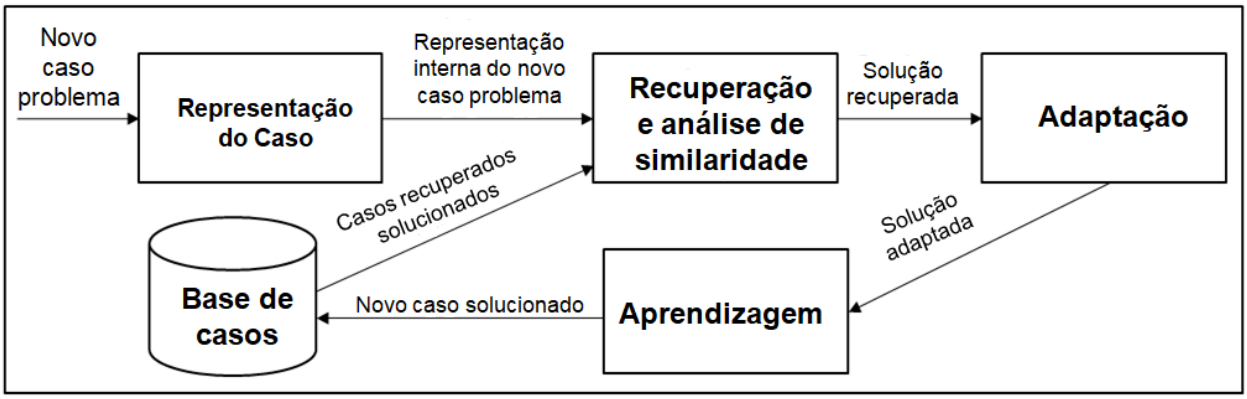


Figura 24. Principais componentes de um sistema RBC. Fonte:Adaptado de [1]

2.6 Aprendizagem de máquina

Aprendizagem de Máquina é uma subárea de Inteligência Artificial cujo objetivo é o desenvolvimento de técnicas computacionais sobre processos de aprendizado [8]. O desenvolvimento dessas técnicas tem possibilitado que as máquinas se tornem capazes de aprender determinadas tarefas.

Mitchell [46] define que "um programa aprende, a partir da experiência *E*, em relação a uma classe de tarefas *T*, com medida de desempenho *P*, se seu desempenho em *T*, medido por *P*, melhora com *E*". Em outras palavras, um programa aprende se seu desempenho melhora a partir da experiência na realização de uma determinada tarefa. Aplicações práticas da aprendizagem de máquina incluem o processamento da linguagem natural, jogos, reconhecimento de voz e imagem, ferramentas de busca e de tradução.

Nessa seção, são apresentadas as principais técnicas de aprendizagem que um agente com aprendizagem utiliza. A seguir são apresentados os três tipos principais de aprendizagem de máquina: aprendizagem supervisionada [25], não supervisionada [46] e por reforço [30].

2.6.1 Aprendizagem supervisionada

O problema da aprendizagem supervisionada envolve a aprendizagem de uma função a partir de suas entradas e saídas [47], isto é, os valores de entrada e saída são informados ao agente e, através de um algoritmo de aprendizagem, esse agente aprende uma função que consegue mapear novos dados de entrada para uma saída correta.

O conjunto dos valores de entrada e saída são chamados de exemplos de treinamento. Normalmente, cada exemplo de treinamento é formado por um vetor de características (entrada) e pelo rótulo da classe (saída). A partir do conjunto de treinamento e de um algoritmo de aprendizagem busca-se aprender um classificador que forneça saídas corretas para novas entradas.

A aprendizagem supervisionada está associada a dois problemas comuns que são a classificação e a regressão. A classificação consiste em atribuir uma instância a uma determinada categoria, ou seja, atribuir um rótulo a cada entrada. Para isso, utiliza-se um classificador aprendido utilizando um conjunto de treinamento, contendo uma entrada e a saída correspondente.

Na classificação, os rótulos assumem valores discretos. Por exemplo, baseado em um conjunto de instancias que representam características de um e-mail, classificá-lo como Notícias, Fórum, SPAM, etc, utilizando para isso um classificador aprendido através de conjunto de treinamento contendo vários exemplos de e-mails com a sua correspondente categoria. Já a regressão consiste em identificar a saída representada por um valor numérico através de uma determinada entrada, a partir de um conjunto de treinamento.

Na Figura 25 está ilustrado um exemplo de regressão, onde cada exemplo de treinamento é o metro quadrado de uma casa em função do seu preço de mercado (50 m² x R$100.000, 100m² x R$200.000, etc). A partir do conjunto de treinamento, é possível que o classificador, faça uma estimativa de quanto custaria, por exemplo, uma casa de 120 m². As desvantagens dessa técnica giram em torno da dependência dos exemplos de treinamento, o que pode gerar a necessidade de muito tempo e esforço para a criação desses exemplos.

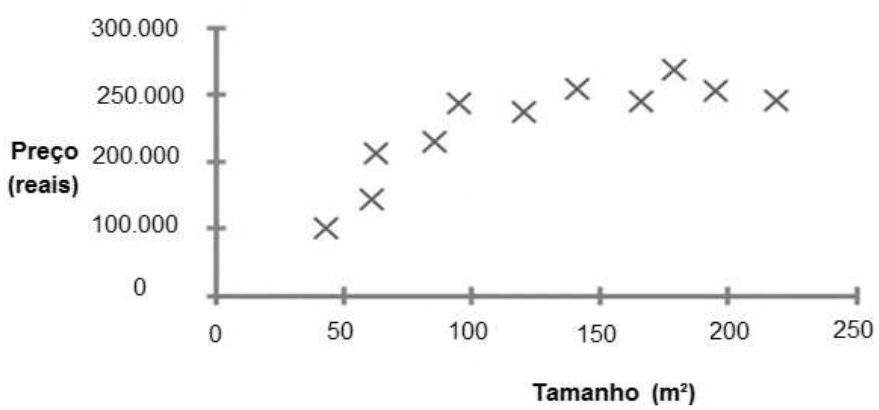


Figura 25. Exemplo de regressão

2.6.2 Aprendizagem não supervisionada

O problema da aprendizagem não supervisionada envolve a aprendizagem de padrões de entrada quando não forem fornecidos valores de saída específicos, ou seja, sem que haja qualquer tipo de treinamento [46][60]. Por exemplo, um agente de táxi poderia aprender quais dias da semana o trânsito possui condições de tráfego bom ou ruim, sem jamais ter recebido exemplos identificados de cada um deles. Isso pode ser feito através do desenvolvimento de "clusters", que são agrupamentos, onde os objetos que pertencem ao cluster são semelhantes uns aos outros e não semelhantes a objetos pertencentes a outros agrupamentos de acordo com algum critério de similaridade. Assim, seriam construídos clusters representando "dias de tráfego bom" e "dias de tráfego ruim". Esses clusters poderiam ter valores de características como "tempo gasto no percurso" e "custo de combustível". Após a construção dos clusters o agente poderia dizer quais dias provavelmente tem bom tráfego. Nem sempre os clusters têm saídas adequadas após a realização do treinamento com o conjunto de dados. Quando isso acontece, é necessário avaliar o motivo pelo qual a saída não está sendo adequada. Algumas causas comuns para a saída dos dados não ser correta é a utilização de exemplos de treinamento insuficientes e a definição de características não relevantes para os objetos que compõem os clusters.

A aprendizagem não supervisionada é comumente utilizada em aplicações de mineração de dados, nas quais se procura identificar padrões em uma grande quantidade de dados para descobrir informações úteis sobre eles.

2.6.3 Aprendizagem por reforço

Na aprendizagem por reforço, o agente aprende a partir de uma série de reforços [60]. Esses reforços são obtidos através da interação do agente com o ambiente e podem ser positivos (recompensa) ou negativos (punição).

Na aprendizagem por reforço não há exemplos de saída correta, como acontece na aprendizagem supervisionada. O reforço obtido através da interação com o ambiente é utilizado para avaliar o comportamento do agente e está associado a um padrão de desempenho que informa se aquele reforço é positivo ou negativo. O aumento do desempenho do agente se dá através da experiência. O fato da interação ser realizada sem treinamento inicial torna a

aprendizagem por reforço atrativa para situações dinâmicas em que é custoso ou difícil reunir um conjunto satisfatório de exemplos de entradas e saídas.

Um exemplo popular de aprendizagem por reforço é o do rato (agente) que se move sobre um labirinto tentando encontrar o queijo. O rato não sabe previamente qual o layout do labirinto (ambiente) ou a localização do queijo. Em cada quadrado, ele deve escolher se mover para cima, para baixo, esquerda ou direita. O "layout" do labirinto é fixo e uma pequena quantidade de queijo é colocada na mesma posição no interior do labirinto. O rato vai explorar o labirinto até encontrar o queijo. Após encontrar o queijo, o rato saberá qual caminho tomou até encontrar o queijo (reforço positivo), mas mesmo assim tentará encontrar um caminho mais curto até o queijo (para maximizar sua medida de desempenho). Depois de várias experiências no labirinto esse rato aprenderá um caminho mais curto a partir do ponto de partida até o queijo. Se depois de algum tempo o queijo for colocado em outro local do labirinto (reforço negativo), o rato irá então reiniciar o processo de busca, procurando o queijo em outros caminhos.

Segundo Richard Sutton et al. [69] um sistema de aprendizagem por reforço possui quatro principais elementos que são: uma política, uma função de recompensa, uma função de valor e um modelo de ambiente, sendo que esse último é opcional.

A política corresponde ao mapeamento das percepções para ação, isto é, dada uma percepção como encontrar a ação apropriada. A política pode ser implementada tanto como condição-ação quanto através de um processo de busca. A política corresponde ao elemento de desempenho do agente com aprendizagem definida por Russel e Norvig.

A função de recompensa define o que é bom ou ruim para o agente e a intensidade, representado por um valor numérico denominado ganho. A função recompensa pode fazer com que a política seja alterada. Por exemplo, se uma dada ação tem uma recompensa baixa, o agente deve escolher outras ações na próxima vez que for realizar a mesma tarefa. A função de recompensa corresponde ao componente crítico do agente com aprendizagem de Russel e Norvig.

A função de valor define o que é bom ou ruim para o agente em longo prazo. Ela indica o ganho total que pode ser acumulado pelo agente a partir de um dado estado. O objetivo do agente é maximizar esse ganho. A função de valor corresponde ao conceito de utilidade de Russel e Norvig.

O modelo do ambiente deve prever o próximo estado e a recompensa de um agente a partir do estado e ação atual. O modelo é considerado opcional, porque nem sempre é possível se representar o ambiente de um agente previamente, é o que acontece em ambientes dinâmicos. Nesses ambientes, vários elementos (objetos ou outros agentes) podem se modificar enquanto o agente está deliberando. Quando o modelo não é fornecido previamente, o agente terá o problema de aprender também como o mundo funciona.

2.7 Trabalhos relacionados

Nas próximas seções são apresentadas as arquiteturas híbridas mais recentes para o desenvolvimento de agentes de software, selecionadas de acordo com a sua relevância para o estado da arte e a sua semelhança com a arquitetura híbrida tese neste trabalho. Além disso, também são apresentados os principais conceitos acerca das arquiteturas de referência, uma vez que a arquitetura híbrida tese neste trabalho é generalizada para uma arquitetura de referência.

2.7.1 Arquiteturas híbridas sem aprendizagem

As arquiteturas híbridas combinam comportamento reativo e deliberativo aproveitando a velocidade de comportamento reativo e a capacidade de raciocínio das arquiteturas deliberativas. Nesta seção serão apresentadas algumas arquiteturas híbridas do estado da arte e em seguida as principais características das mesmas são comparadas

2.7.1.1 Arquitetura híbrida definida por Qinzhou e Lei

A arquitetura definida por Qinzhou e Lei [54], ilustrada na Figura 26, é uma arquitetura de agente híbrido que usa a abordagem RBC. A arquitetura é composta por oito módulos: um módulo de conhecimento encarregado de armazenar a base de casos e o conjunto de regras; um módulo condição-ação responsável pelo comportamento reativo; um módulo de percepções encarregado de receber informações do ambiente; um módulo de aprendizagem que utiliza algoritmos de aprendizagem RBC para aumentar a eficiência da recuperação dos casos; um módulo de recuperação, cuja principal função é comparar um novo caso com um caso antigo da base de casos e realizar a análise de similaridade entre eles; um módulo de decisão, que corresponde ao modulo deliberativo, responsável por realizar um processo de raciocínio sobre os casos utilizando o conjunto de regras do modulo de conhecimento; um módulo de execução que é responsável por realizar ações no ambiente; um módulo de comunicação que é encarregado das interações entre os agentes e que para isso utiliza a linguagem KQML [79].

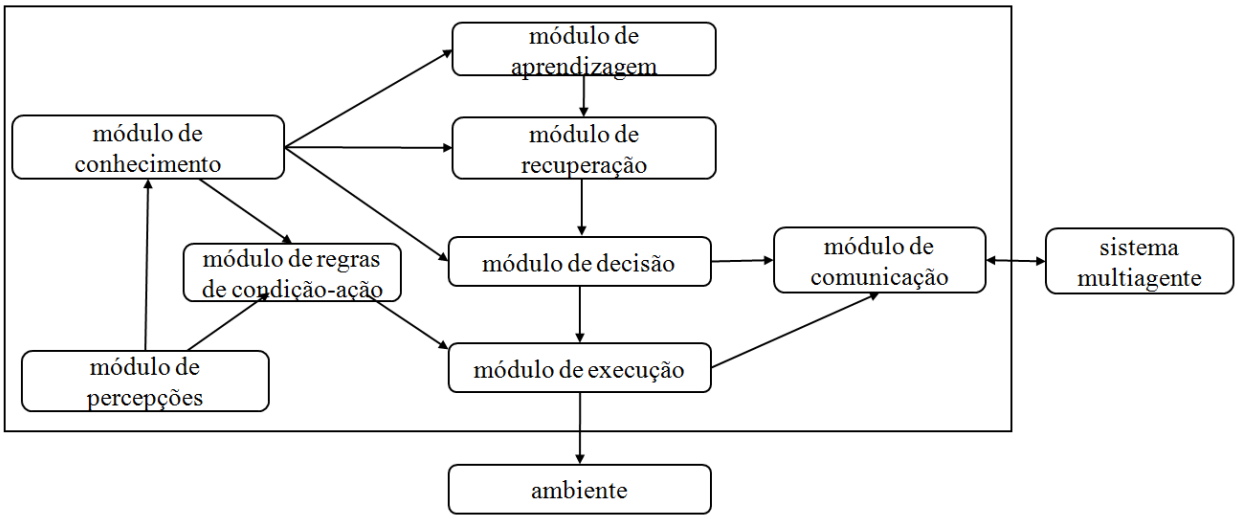


Figura 26. Arquitetura híbrida definida por Qinzhou e Lei. Fonte: Adaptado de [54]

2.7.1.2 Arquitetura híbrida definida por Sun e Wu

Uma arquitetura hibrida que combina uma arquitetura reativa e outra deliberativa BDI ("Beliefs, Desire and Intentions") é definida por Sun e Wu [68]. Esta arquitetura está representada na Figura 27. A camada reativa da arquitetura é responsável pelos comportamentos mais elementares do agente enquanto a camada BDI pelos comportamentos mais complexos. O agente escolhe utilizar a arquitetura reativa ou BDI dependendo da medida de confiança, previamente inserida na base de conhecimento do agente, na ação a ser executada. Se a medida de confiança for alta o agente utilizará a arquitetura reativa e se ela for baixa a arquitetura deliberativa será utilizada.

A abordagem utilizada na arquitetura híbrida de Sun e Wu para escolha do comportamento, se reativo ou BDI, torna as ações do agente mais efetivas devido a possibilidade de selecionar ações com altos níveis de confiança definidos em tempo de projeto (maior probabilidade der ter resultados satisfatórios). No entanto, definir o nível de confiança de cada ação pode exigir muito esforço do desenvolvedor, uma vez que a arquitetura não contempla nenhuma forma de aprendizagem automática.

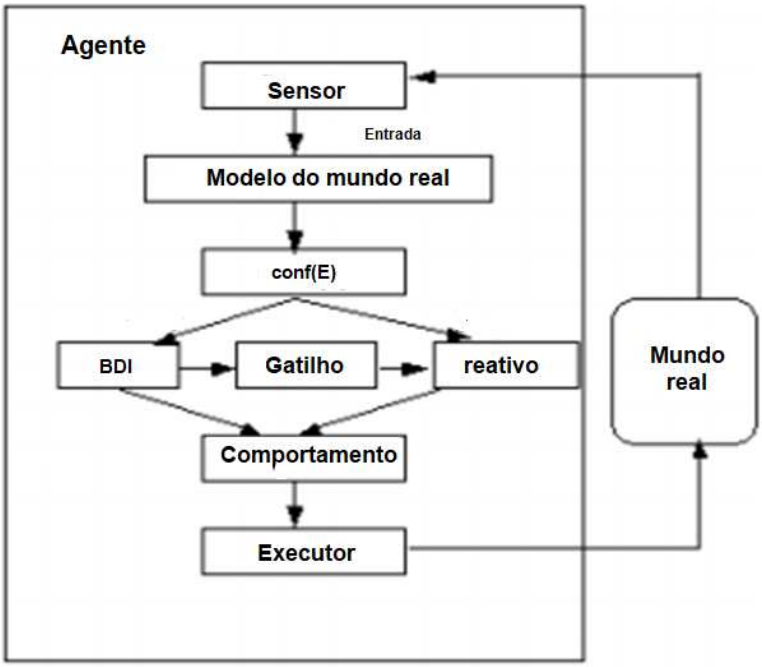


Figura 27. Arquitetura híbrida definida por Y. Sun e B. Wu. Fonte: [68]

2.7.2 Arquiteturas híbridas com aprendizagem

Uma característica comum à maioria das arquiteturas híbridas atuais é que elas são também cognitivas. Uma arquitetura cognitiva especifica a infraestrutura básica para o desenvolvimento de um sistema inteligente [36]. Esse tipo de arquitetura tem por objetivo desenvolver agentes capazes de atuar utilizando fenômenos cognitivos humanos como memorização, aprendizagem, tomada de decisão, processamento da linguagem natural, etc. Assim, a principal diferença entre as arquiteturas híbridas cognitivas e as arquiteturas híbridas apresentadas anteriormente é que as arquiteturas cognitivas são mais ambiciosas no seu escopo tentando emular a maioria dos fenômenos cognitivos humanos, enquanto que as demais arquiteturas híbridas se limitam a determinados comportamentos inteligentes como o raciocínio, comunicação e ações reflexas.

As arquiteturas cognitivas geralmente incluem [36]: memória de curto e longo prazo, um mecanismo de desempenho e um mecanismo de aprendizagem.

Nas seções seguintes, quatro arquiteturas cognitivas híbridas atuais são apresentadas: SOAR [34][35], ACT-R (Adaptive Control of Thought–Rational) [5], InteRRaP (Integration of Reactive Behavior and Rational Planning) [49] e CLARION [39][67].

2.7.2.1 Arquitetura SOAR

SOAR é uma das arquiteturas híbridas mais completas, e vem sendo aperfeiçoada desde 1983 [34][35]. SOAR possui um ambiente de desenvolvimento e um framework para o suporte à criação de agentes conforme suas definições. É, também, uma arquitetura cognitiva dirigida por objetivos que integra o raciocínio, execução reativa, planejamento e diversas técnicas de aprendizagem com o objetivo de criar um sistema computacional geral que possua as mesmas habilidades cognitivas que os seres humanos. No entanto, o objetivo de SOAR ainda não foi alcançado, uma vez que os sistemas cognitivos inteligentes ainda não conseguiram ser tão gerais quanto a capacidade cognitiva humana, de modo que esse projeto consegue simular, até o momento corrente, a cognição apenas em algumas tarefas específicas.

A última versão disponível da arquitetura SOAR (versão 9) durante a escrita deste trabalho está ilustrada na Figura 28. A arquitetura é composta pela memória de trabalho, também conhecida por memória de curto prazo, pela memória de longo prazo, por um módulo de raciocínio, por módulos responsáveis por gerenciar as percepções e ações dos agentes e também por módulos responsáveis pela aprendizagem do agente.

Na Figura 28, pode-se notar que SOAR faz uma distinção entre memória de longo prazo (Perceptual STM Memory – Short Time Memory) e memória de curto prazo (Perceptual LT Memory – Long Term Memory) já durante as suas percepções e também possui um módulo para representação de imagens (Mental Imagery).

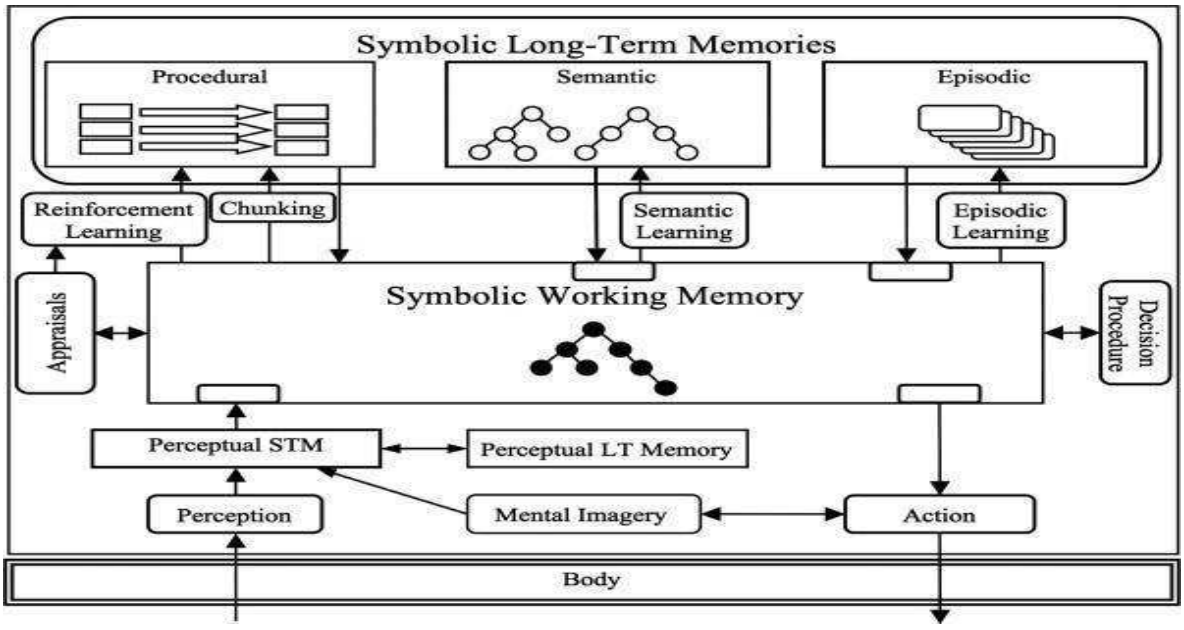


Figura 28. Arquitetura SOAR. Fonte: [22]

A memória de trabalho de SOAR representa todas as informações dinâmicas sobre o agente. Na arquitetura SOAR, toda a memória de trabalho é organizada como estruturas de gráfico de estados. Assim, cada elemento da memória de trabalho está ligada direta ou indiretamente a um símbolo do estado. Na Figura 29 está um exemplo simples de como a estrutura da memória funciona. No exemplo, são representados dois blocos A e B sobre uma mesa. Na Figura 30 está ilustrada a memória de trabalho do agente utilizada para representar os blocos A e B.

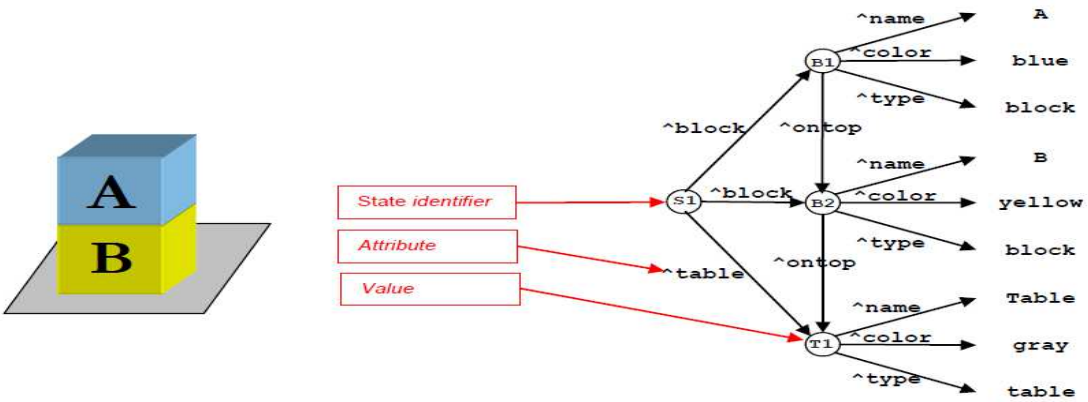


Figura 29. Estrutura da Memória de Trabalho de SOAR. Fonte: [34]

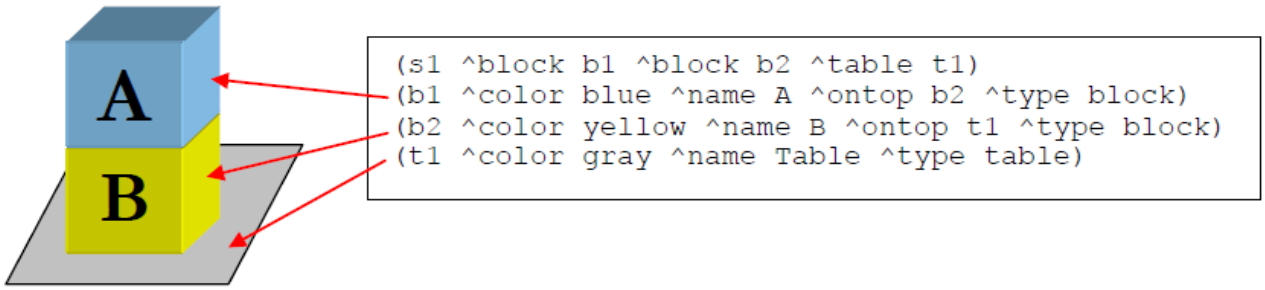


Figura 30. Representação da memória de trabalho de um agente SOAR. Fonte: [34]

Uma das formas de representar a memória de longo prazo no SOAR é através de regras de condição-ação. Essas regras têm a estrutura do exemplo ilustrado na Figura 31. Por definição toda regra sempre começa com a sigla "sp" que signitifica "Soar production", seguida do nome da regra, da condição e ação associada.

```
sp {hello-world
   (state <s> ^type state)
-->
(write |Hello World|)
(halt)}
```

Figura 31. Regra condição-ação SOAR. Fonte: [34]

SOAR suporta várias formas de aprendizagem tais como aprendizagem por reforço, aprendizagem episódica [46], aprendizagem semântica [46] e "chunking" [35]. Na Figura 32 está exemplificada a definição de uma regra simples para a aprendizagem por reforço. Na regra está representado que se o agente mover-se para a esquerda receberá um reforço negativo (penúltima linha) e mover-se para a direita um reforço positivo.

```
sp {propose*initialize-left-right
(state <s> ^superstate nil
-^name)
-->
(<s> ^operator <o> +)
(<o> ^name initialize-left-right)
}
sp {apply*initialize-left-right
(state <s> ^operator <op>)
(<op> ^name initialize-left-right)
-->
(<s> ^name left-right
^direction <d1> <d2>
^location start)
(<d1> ^name left ^reward -1)
(<d2> ^name right ^reward 1)
}
```

Figura 32. Exemplo da definição de reforço na arquitetura SOAR. Fonte: [34]

2.7.2.2 Arquitetura ACT-R

ACT-R é uma arquitetura cognitiva híbrida que tenta emular os processos de cognição humano como o aprendizado e a aquisição de conhecimento através de um sistema de produção [5][70]. A arquitetura híbrida ACT-R, ilustrada na Figura 33, é composta por uma camada de percepção/motora ("Perceptual/Motor Layer"), uma camada de cognição ("Cognition Layer") e uma camada intermediaria denominada de "ACT-R buffers". Na camada perceptual/motora, a arquitetura emula os canais de processamento de informação de entrada e saída humanos através dos módulos visual, motor, fala e audição. Na camada de cognição, a arquitetura representa a memória do agente através de dois tipos de conhecimento declarativo e memória de produção. A memória declarativa corresponde à parte reativa do sistema e é formada por "chunks", sendo que um chunk é uma estrutura de atributo-valor, com um tipo especial de atributo denominado "ISA" que determina o tipo do "chunk". A memória de produção é formada por regras de condição-ação e um motor de inferência para selecionar as ações na base de conhecimento. O módulo

intermediário “ACT-R Buffers” é a uma memória de trabalho do agente, corresponde ao conhecimento utilizado somente no momento de realizar uma determinada tarefa. Esse conhecimento pode ser recuperado tanto da memória declarativa quanto da memória de produção. A aprendizagem na arquitetura ACT-R é por “chunking”, que consiste, basicamente, no aprendizado de pedaços úteis de informação (“chunks”) que são armazenados na memória declarativa para uso futuro. ACT-R possui um ambiente de programação que suporta o desenvolvimento de agentes híbridos segundo suas definições.

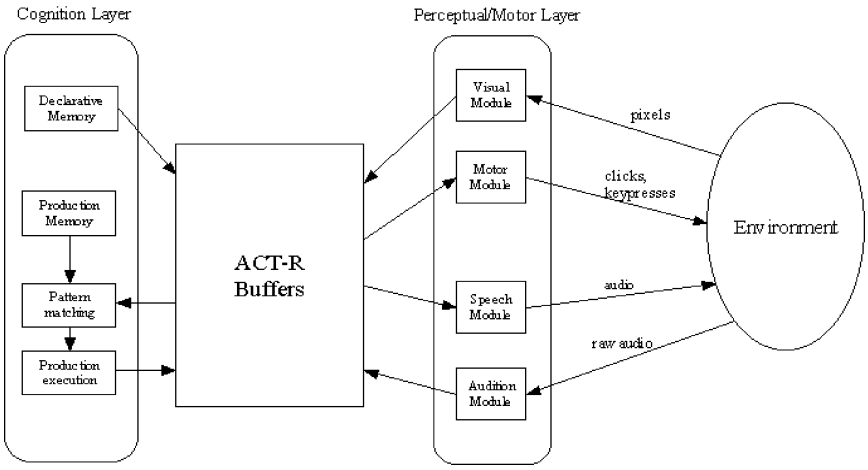


Figura 33. Arquitetura ACT-R. Fonte: [5]

2.7.2.3 Arquitetura InteRRaP

A arquitetura InteRRaP é uma típica arquitetura híbrida de duas passagens que suporta o desenvolvimento de agentes reativos e de agentes dirigidos por objetivos. Ela é organizada em camadas unidas por uma estrutura de controle e uma base de conhecimento associada a cada camada [49].

A arquitetura InteRRaP, ilustrada na Figura 34, consiste basicamente de cinco módulos: a interface com o ambiente (“World interface”), o componente baseado em comportamento (“Behavior-based Layer”), o componente baseado em planos (“Plan-Based Layer”), o componente de cooperação (“Cooperation Layer”), e a base de conhecimento do agente. A interface com o ambiente é responsável pela percepção, ação e comunicação do agente. O componente baseado em comportamento implementa o comportamento reativo e o conhecimento procedural do agente. O componente baseado em planos contém mecanismos de planejamento, que concebe planos de agentes isolados. Finalmente o componente de cooperação contém um mecanismo para criar a união desses planos.

A base de conhecimento de InteRRaP é formada por três camadas. A camada mais baixa contém fatos que representam o modelo do mundo do agente bem como representações de ações e padrões de comportamento. A segunda camada contém o modelo mental do agente. A terceira camada é composta do modelo social do agente, que fornece, por exemplo, conhecimentos de estratégias para cooperação e crenças sobre metas de outros agentes.

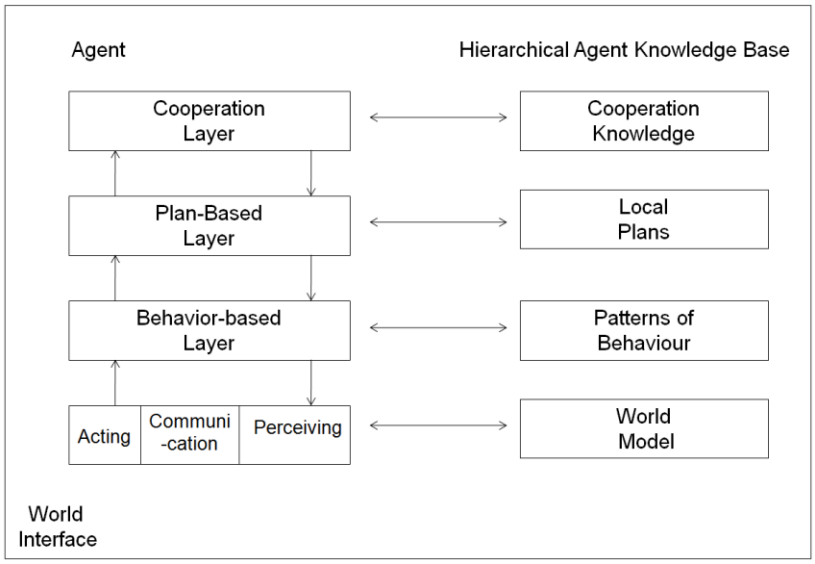


Figura 34. Arquitetura INTERRAP. Fonte: Adaptado de [49]

2.7.2.4 Arquitetura CLARION

A arquitetura CLARION ("Connectionist Learning with Adaptive Rule Induction Online") [39][67] é uma arquitetura híbrida dirigida por objetivos. CLARION é uma arquitetura integrada, composta por um conjunto de subsistemas distintos, com uma estrutura de representação do conhecimento dupla em cada subsistema (representações implícitas e explícitas).

A arquitetura CLARION é ilustrada na Figura 35. Ela é composta de 4 módulos básicos: o módulo *Action Centred* Subsystem (ACS), responsável pelo comportamento reativo (explicit representation) e deliberativo (implicit representation); o módulo *Non Action Centred* Subsystem (NACS) que representa a base de conhecimento do agente, que, por sua vez, é dividida em dois módulos um para o conhecimento reativo (regras) e outro para o conhecimento deliberativo (redes neurais); o módulo Motivacional Subsystem (MS) é responsável pelos objetivos do agente e o módulo Meta-cognitive Subsystem é responsável pelo aprendizado do agente utilizando a técnica de aprendizagem por reforço.

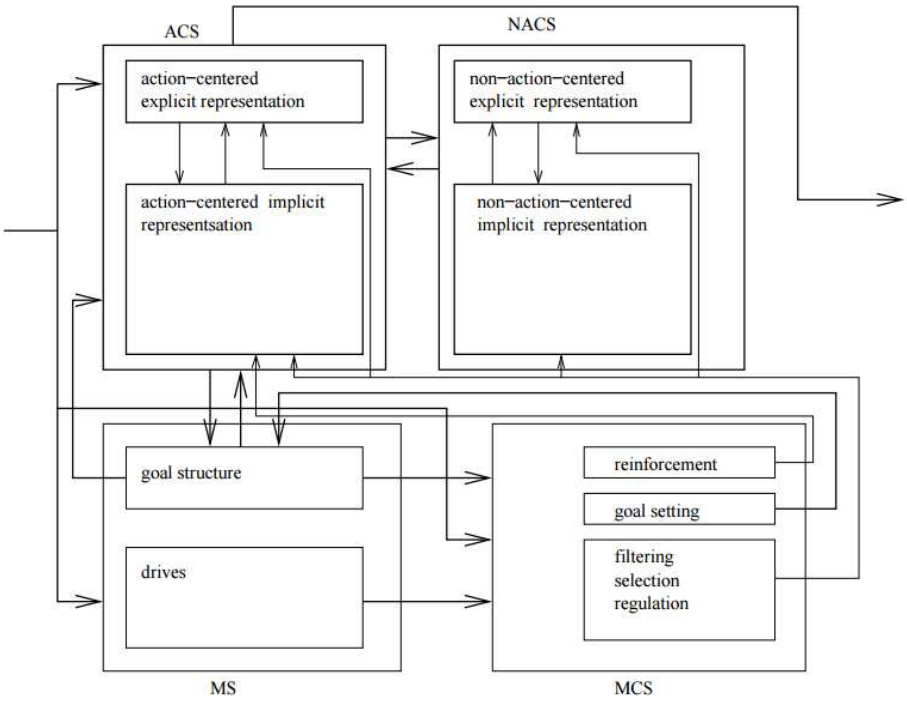


Figura 35. A arquitetura CLARION. Fonte: Sun [67]

2.7.3 Estudo comparativo

Na Tabela 2 as principais características das arquiteturas híbridas definidas por Qinzhou e Lei e por Sun e Wu e as arquiteturas híbridas cognitivas SOAR, ACT-R, INTERRAP e CLARION são apresentadas.

A arquitetura de Qinzhou e Lei têm como principal característica utilizar raciocínio baseado em casos e aprendizagem não-supervisionada para encontrar uma solução a uma dada percepção. Infelizmente não são relatadas na literatura aplicações desta arquitetura que permitam a sua avaliação. O que distingue a arquitetura definida por Sun e Wu é o seu mecanismo de confiança para as ações, isto é, o agente escolhe qual comportamento utilizar, reativo ou deliberativo, dependendo do nível de confiança das ações da base de conhecimento. No entanto, a atribuição de níveis de confiança para as ações exige muito esforço do desenvolvedor e em determinados domínios pode se tornar inviável. Dentre todas as arquiteturas apresentadas, a mais documentada na literatura é a arquitetura SOAR que inclui diferentes formas de aprendizagem, representação da base de conhecimento do agente, além de uma linguagem própria para desenvolvimento de agentes. SOAR também fornece ferramentas para desenvolvimento dos agentes, documentação atualizada e exemplos de agentes implementados para problemas diversos. A arquitetura CLARION, assim como a SOAR, também é bem documentada e se

destaca pela boa definição e separação de seus módulos e pela separação do conhecimento em explícito e implícito o que facilita o desenvolvimento de agentes de software híbridos. A arquitetura ACT-R difere da arquitetura SOAR e da arquitetura CLARION por ser mais restrita em relação as alternativas de representação da base de conhecimento do agente e da técnica de aprendizagem utilizada. No entanto, essa arquitetura possui um tratamento das percepções mais amplo, incluindo representação de imagens e da voz. INTERRAP, por sua vez, tem como principal vantagem em relação às demais arquiteturas híbridas, a sua organização em camadas, uma vez que a definição de vários componentes independentes insere complexidade na arquitetura, pois é necessário gerenciar as interações entre esses componentes.

Tabela 2. Quadro comparativo entre arquiteturas híbridas

Características / **Arquiteturas**	**Componente Reativo**	**Componente Deliberativo**	**Técnica de aprendizagem**	**Representação da base de conhecimento**
Arquitetura definida por Qinzhou e Lei	Reativa sem estado	RBC	Aprendizagem não supervisionada	Base de Casos
Arquitetura definida por Sun e Wu	Reativa sem estados	Raciocínio prático (abordagem BDI)	Não possui aprendizagem	Conhecimento procedural
Arquitetura SOAR	Reativa com estado	Raciocínio dedutivo (regras de produção)	Aprendizagem por reforço, episódica, “chunking” e semântica	Regras de condição-ação, grafo de estados, semântica e episódios.
Arquitetura CLARION	Reativa com estado	Dedutivo /Sistema de produção	Aprendizagem por reforço	Regras e redes neurais
Arquitetura ACT-R	Reativa com estado	Raciocínio dedutivo (regras de produção)	Aprendizagem por “chunking”.	Regras, fatos e conhecimento procedural.
Arquitetura INTERRAP	Reativa com estado	Raciocínio prático (abordagem BDI)	Não possui aprendizagem	Conhecimento procedural

2.7.4 A Segurança da Informação como domínio de avaliação da tese

A segurança da informação é um conjunto de estratégias que tem por objetivo proteger os ativos de informação e pode incluir ferramentas e políticas para prevenir e detectar ameaças que afetem a segurança dos dados de acesso não autorizado [27].

Os princípios básicos da segurança da informação são confidencialidade, integridade e a disponibilidade. A confidencialidade é a propriedade de que a informação não será disponibilizada ou divulgada a indivíduos, entidades ou processos sem autorização. A integridade é a propriedade de proteção à precisão e perfeição de ativos. A disponibilidade é a propriedade de que a informação esteja acessível e utilizável sob demanda por uma entidade autorizada [2]. Outra característica importante é que a segurança da informação visa garantir a autenticidade da informação. Este conceito está relacionado com o princípio de integridade da informação, uma vez que um dos requisitos para a informação ser autentica é que o seu conteúdo não tenha sido alterado por terceiros. Além disso, para que a informação seja considerada autêntica, outras propriedades, como autor, origem e data de criação devem ser verdadeiras.

Segundo Whitman [77] há 12 tipos de ameaças mais comuns que podem afetar os princípios básicos da segurança da informação:

1. Erro humano. Por exemplo, alterando um arquivo acidentalmente.
2. Comprometimento da propriedade intelectual. Por exemplo, a utilização de software não licenciado;
3. Atos de espionagem. Por exemplo, coletando dados sem autorização;
4. Atos de extorsão. Por exemplo, ameaçando divulgação de informação privada;
5. Atos de vandalismo. Por exemplo, destruindo ou alterando informações por diversão;
6. Atos de roubo. Por exemplo, roubando informações ou equipamentos.
7. Ataque por software. Por exemplo, atacando o sistema alvo usando algum tipo de vírus.
8. Forças da natureza que destroem equipamentos ou informações como incêndio, terremoto, etc.
9. Mudança na qualidade de serviço. Por exemplo, diminuição da velocidade do link de Internet pelo provedor de acesso devido a problemas técnicos.

10. Falhas de hardware. Quando algum equipamento deixa de funcionar adequadamente.
11. Falhas de software. Por exemplo, erros causados por problemas de codificação.
12. Obsolescência técnica. Por exemplo, utilizando tecnologias ultrapassadas e que são vulneráveis a ataques.

Para garantir os princípios de segurança e evitar que softwares maliciosos invadam um computador, foram desenvolvidos mecanismos de segurança como criptografia, backups, ferramentas antimalware (antivírus, antispyware, antispam etc.), firewall e sistemas de detecção de intrusão (IDSs).

Um antivírus é um software que monitora o sistema operacional ou uma rede para detectar a presença de vírus. Ele, normalmente, possui um banco de dados contendo um conjunto de regras para detecção de vírus que é atualizado periodicamente.

Um firewall analisa o tráfego de uma rede e identifica se os dados são confiáveis (de fontes confiáveis, não alterados, etc) através de um conjunto de regras armazenadas em um banco de dados, essas regras podem ser tanto criadas pelo desenvolvedor do software quanto pelo administrador da rede. Aqueles dados que não atendem as regras são impedidos de trafegar na rede pelo firewall.

Um antispyware é um software similar a um ativírus, ele escaneia o computador ou rede em busca de software espiões como o keylloger que captura dados digitados pelo usuário. Muitas vezes o antispyware vem integrado ao software antivírus. Ele é importante para evitar que informações sensíveis dos usuários como dados bancários sejam utilizados por pessoas mal-intencionadas.

Um anti-spam é um software que evita que mensagens não solicitadas sejam enviadas aos usuários, evitando, por exemplo, o excesso de informação. Ele funciona baseado em filtros que são criados pelos próprios usuários, onde são explicitados quais tipos de mensagens (remetente, palavras-chaves, etc) são indesejáveis e devem ser descartadas ou por filtros criados pelos próprios provedores de e-mail.

Segundo Benmoussa [6], a detecção de intrusão é o processo de monitoramento dos eventos que ocorrem em um sistema de computador ou rede, analisando-os para sinais de

intrusões. Um IDS pode detectar invasões a um sistema de computação e alertar o administrador da rede para que ele possa tomar as ações corretivas para evitar a intrusão.

Os IDSs realizam o monitoramento de uma rede ou de um dispositivo conectado a ela com o objetivo de encontrar a ocorrência de algum ataque ou atividade maliciosa. Os IDS que monitoram uma rede são chamados de "Network based Intrusion Detection System" (NIDS) e os que monitoram apenas um dispositivo são chamados de "Host based Intrusion Detection System" (HIDS). Um NIDS é geralmente instalado em uma fronteira entre redes e é muito útil para detectar acessos não autorizados externos à rede. Um HIDS monitora o tráfego do host onde ele está instalado, logs de aplicativos, modificação de acesso de arquivos, modificações de configurações do sistema operacional, processos em execução e atividades de aplicações. É instalado em dispositivos conectados a uma rede como servidores e computadores pessoais.

Existem duas abordagens principais para implementação de um IDS: baseado em anomalia ou baseado em assinatura. Um IDS por anomalia é uma abordagem usada para identificar um comportamento anormal de um host ou uma rede [4]. A abordagem por assinatura se baseia em um conjunto predefinido de assinaturas de ataques, os quais são listados em uma base de dados em forma de regras de detecção [6].

As principais funções de um IDS são [6]: examinar o tráfego de uma rede, identificar possíveis eventos monitorando o usuário e o sistema, registrar informações sobre o usuário do sistema, analisar configurações e vulnerabilidades do sistema, avaliar a integridade dos arquivos do sistema, reconhecer atividades e padrões típicos de ataques e enviar um alerta para o administrador da rede.

O domínio de segurança da informação, mais precisamente a detecção de intrusão a redes de computadores foi escolhido como domínio de aplicação do agente híbrido com aprendizagem desenvolvido por duas características principais: a *necessidade de aprendizagem*, uma vez que os mesmos ataques a uma rede de computadores se repetem, é relevante que o agente de software aprenda regras para esses ataques de modo a ser mais eficiente nas suas próximas ações; e a *proximidade do conhecimento de ataques a redes de computadores ao RBC, similar a estrutura de um caso*, isto é, muitas vezes não se tem todo o conhecimento de um determinado ataque, mas normalmente, se tem a informação básica do problema do ataque e da sua solução, da mesma forma que o RBC funciona.

2.8 Síntese

Neste capítulo, a fundamentação teórica básica para o entendimento da arquitetura de híbrida com aprendizagem tese foi apresentada, desde o conceito de agentes de software até as arquiteturas híbridas atuais. Nas primeiras seções, foram introduzidos os tipos básicos de agentes de software com suas principais propriedades, bem como uma correspondência entre as terminologias utilizadas para diferenciar os agentes conforme suas características e os componentes internos de um agente genérico. Em seguida, foram apresentadas as principais técnicas de aprendizagem, as arquiteturas híbridas atuais e uma visão geral do domínio de aplicação a ser utilizado nos estudos de casos.

3. HyLAA: UMA ARQUITETURA HÍBRIDA COM APRENDIZAGEM PARA O DESENVOLVIMENTO DE AGENTES DE SOFTWARE

Neste capítulo, a arquitetura híbrida com aprendizagem HyLAA ("Hybrid Learning Agent Architecture") é apresentada. Para avaliá-la, um agente híbrido foi desenvolvido aplicado ao domínio da Segurança da Informação, mais precisamente, à detecção de intrusão em redes de computadores, já definido no capítulo de fundamentação teórica deste manuscrito. Para a base de conhecimento do agente, reusou-se a ONTOID [44], uma ontologia para segurança da informação e para detecção de intrusão em rede de computadores.

O processo MADAE-Pro ("Multi-agent Domain and Application Engineering Process") [18][19] [38] que recomenda a construção de um conjunto de modelos para as fases de análise, projeto e implementação de um agente de software foi utilizado como guia para o desenvolvimento do agente HyLAA. No entanto, a fase de projeto do agente do MADAE-Pro foi adaptada com as diretrizes da arquitetura HyLARA, uma vez que o MADAE-Pro original ainda não suporta o desenvolvimento de agentes híbridos. Assim, o MADAE-Pro será estendido para contemplar o desenvolvimento de agentes híbridos através da experiência obtida no desenvolvimento do agente híbrido apresentado nesse capítulo. O ambiente de desenvolvimento de agentes de software MADAE-IDE [10] [11] também não suporta ainda o desenvolvimento de agentes híbridos. Por isso, utilizamos a ferramenta ONTORMAS [37] para criação dos modelos. A ONTORMAS é uma ferramenta com a qual se constrói os modelos de forma manual e o ambiente MADAE-IDE de forma semi-automatizada.

Dois modelos do MADAE-Pro usados para representar os requisitos de uma aplicação multiagente são o Modelo de Conceitos e o Modelo de Objetivos (Figura 36 e Figura 37), utilizados, respectivamente, para representar os principais conceitos do domínio de aplicação e para definir os objetivos que o agente ou sistema multiagente deverá atingir.

No Modelo de Conceitos foram definidos os principais conceitos do domínio da Segurança da Informação e dos IDSs, por exemplo, o conceito "System to detect threats to information security" possui os relacionamentos "identifies", "monitors", "uses", "performed", "provides" e "analyzes" com os conceitos "Data package", "Computer network", "ONTOID", "Corrective solutions", "Solutions" e "Monitored Package". Esses conceitos são o primeiro rascunho da base de conhecimento do agente, e são construídos com a orientação de um especialista no domínio. Na fase de projeto do agente, esses conceitos são refinados e mapeados para uma ontologia.

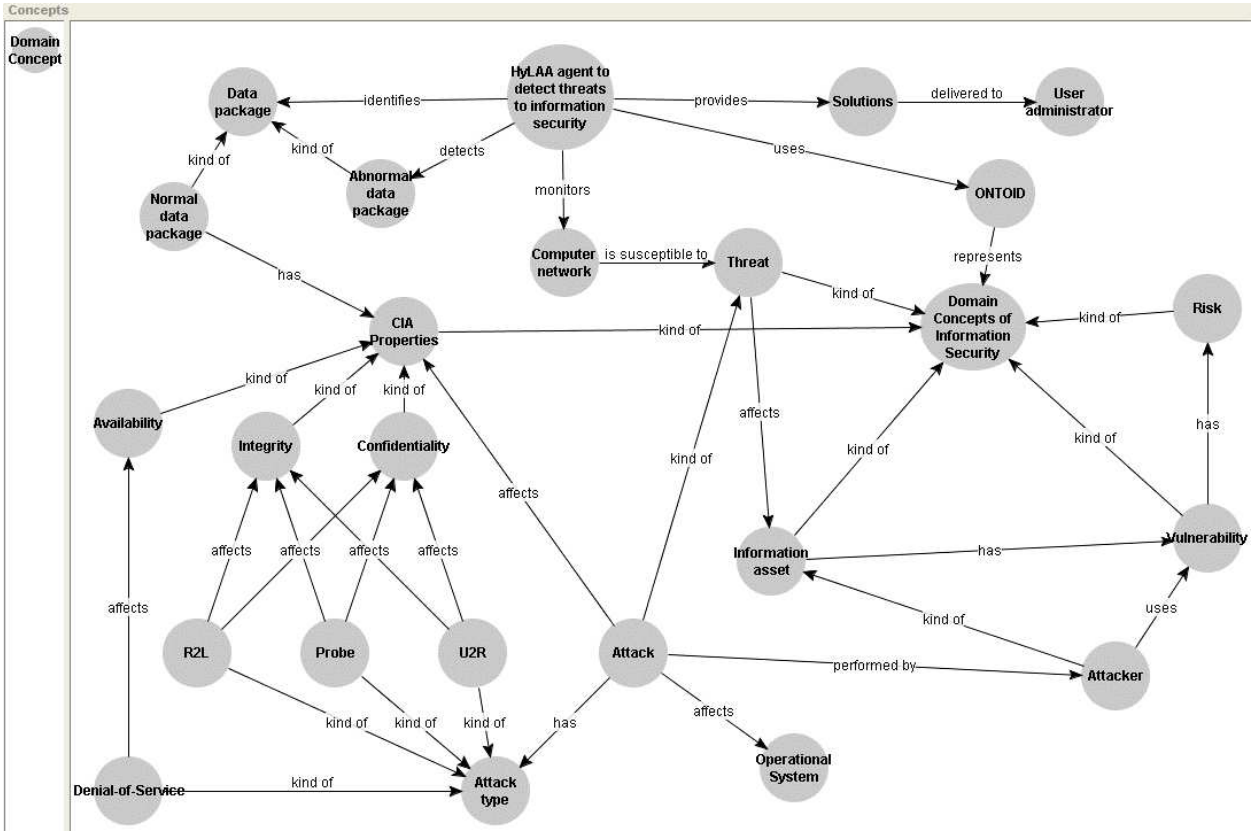


Figura 36. Modelo de conceitos do agente híbrido HyLAA

No Modelo de Objetivos (Figura 37) foram definidos os objetivos geral e específicos do agente bem como as responsabilidades que são necessárias para que cada objetivo específico seja alcançado e a entidade externa, com a qual o agente troca informações. Para alcançar o objetivo geral que é prover uma solução baseada em casos para segurança da informação (“provide case-based solution to information security intrusions”), é necessário o objetivo específico manter uma base de conhecimento contendo casos de intrusão (“Maintain a knowledge base containing cases of intrusion”) que possui as responsabilidades de armazenar casos (“store cases”) e atualizar a ontologia (“update ontology”); o objetivo específico monitorar redes de computadores (“monitor computer network”) com a responsabilidade de obter informações sobre um pacote de dados (“obtain information about a data package”); o objetivo específico detectar intrusões (“detect intrusions”), tendo como responsabilidades criar uma instancia de uma caso problema na ontologia (“create an instance of a case problem in the ontology”) e analise de similaridade (“similarity analysis”); e o objetivo específico entregar uma solução para as intrusões identificadas (“delivery solution to identified intrusions”) com a responsabilidade de escolher a

solução mais similar ao caso problema (“choice more similar solution for the case”). O modelo de objetivos irá guiar todo o desenvolvimento do agente.

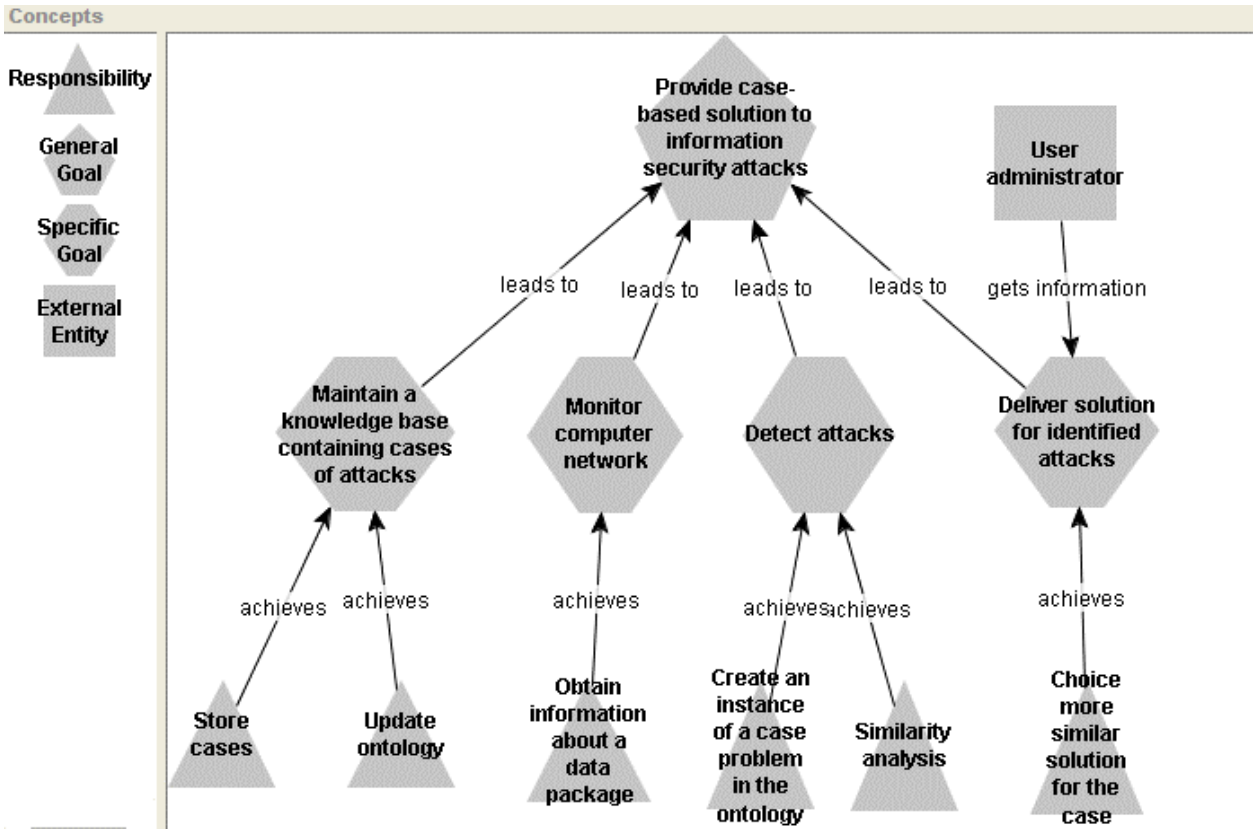


Figura 37. Modelo de objetivos do agente híbrido RBC com aprendizagem

3.1 Visão geral da arquitetura HyLAA

A Figura 38 ilustra a arquitetura do agente HyLAA. Ela possui dois componentes principais: o desempenho (“Performance”) e a aprendizagem (“Learning”). O componente de desempenho tem quatro subcomponentes: interpretação da percepção, sistema reativo, ONTOID e o sistema RBC (“CBR System”). O subcomponente interpretação da percepção (“Perception Interpretation”) é responsável por identificar o tipo de percepção: uma nova intrusão ou um realimentação sobre o sucesso ou fracasso de uma ação de detecção de intrusão anteriormente executada. ONTOID é uma ontologia utilizada como a base de conhecimento do agente compartilhada com os sistemas reativo (“Reactive System”) e do sistema RBC (“CBR System”). O sistema reativo é responsável por executar as regras de condição-ação, comparando cada percepção do agente com as regras da base de conhecimento do agente. O sistema RBC busca fazer a recomendação de uma solução similar para uma determinada intrusão. A entrada do sistema RBC é uma percepção que descreve uma nova intrusão identificada. A partir desta

percepção, uma representação interna é criada, cujos valores de atributos são comparados com os casos de intrusões disponíveis na ONTOID, a fim de identificar uma intrusão semelhante. Quando o sistema RBC encontra uma invasão semelhante na ONTOID, sua solução é reutilizada e, eventualmente, adaptada.

O componente de aprendizagem ("Learning") é responsável por fazer melhorias no comportamento do agente através da aprendizagem supervisionada. O componente crítico ("Critic") informa ao sistema de aprendizagem sobre o sucesso do agente de acordo com um padrão fixo de desempenho ("Performance standard"). Ele usa o realimentação do crítico sobre os resultados das ações dos agentes e determina como o sistema de desempenho deve ser modificado para ser melhor no futuro. Para isso, o agente cria gradualmente um conjunto de treinamento com cada par <percepção, ação> que atende o padrão de desempenho. Quando o conjunto de treinamento tem uma série de exemplos considerados suficientes para a aprendizagem, o subcomponente de treinamento gera um classificador que contém um conjunto de regras reativas que está incluído na base de conhecimento do agente ONTOID e atualiza a base de conhecimento com as novas regras aprendidas. Consequentemente, o desempenho do agente em geral será melhorado.

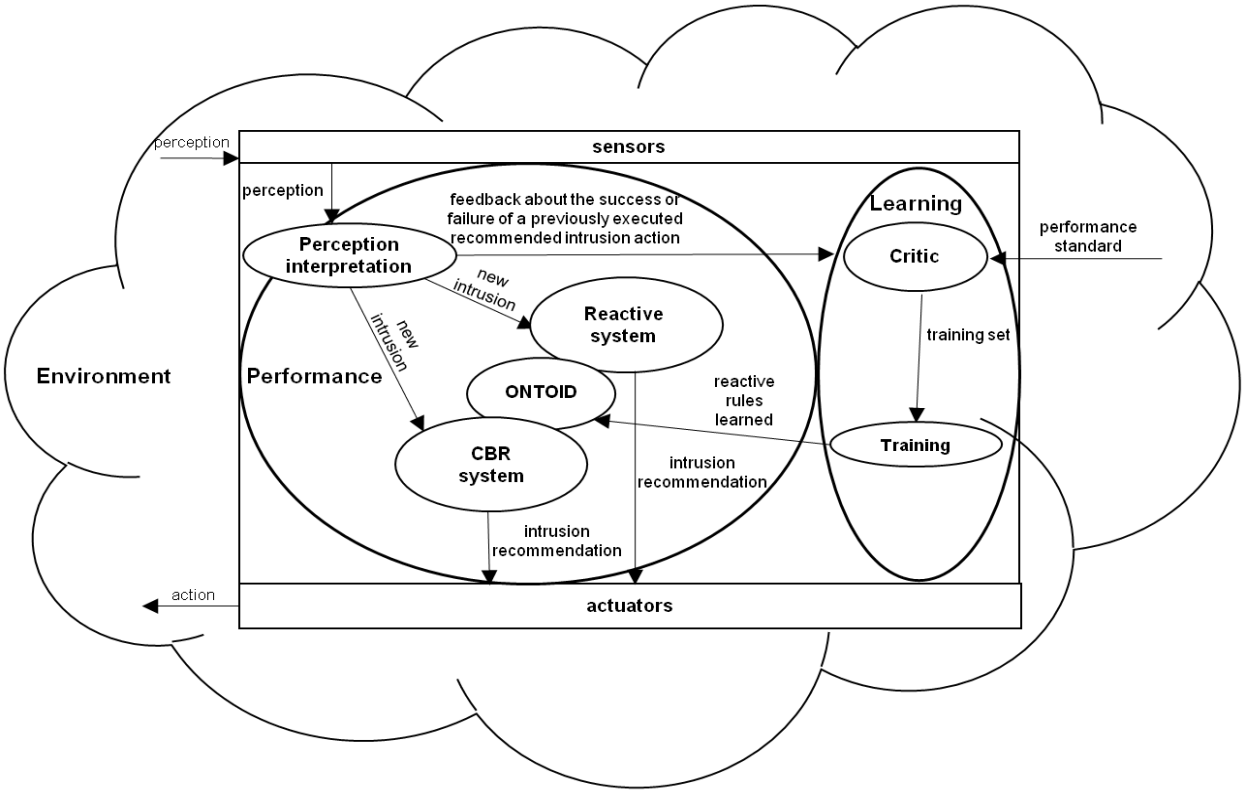


Figura 38. Arquitetura do agente HyLAA

A Figura 39 ilustra o diagrama de estados genéricos por que passa o agente HyLAA, ou seja, uma visão dinâmica. O estado inicial do agente é “Monitoring”. Neste estado o agente monitora a rede à espera de uma percepção que pode ser tanto uma nova percepção quanto o resultado de uma ação já executada no ambiente (“realimentação”).

Quando o agente percebe uma nova intrusão, um novo caso problema é instanciado e o agente entra no estado “Processing for a reactive solution”. No caso de encontrar uma ação reativa para essa percepção, a ação é executada no ambiente e ele volta ao estado “Monitoring”, mas se não encontrar ele entra no estado “Processing for a case-based solution”. Neste estado, ele calcula a similaridade entre o novo caso-problema e os casos da base de conhecimento. Essa ação é executada no ambiente e ele volta ao estado “Monitoring”.

Quando o agente tem uma percepção com o resultado de uma ação executada no ambiente, entra no estado “Processing successful behaviours”, no qual o agente avalia se a ação teve bom resultado ao ser executada no ambiente e, caso tenha, armazena para compor o conjunto de treinamento que será utilizado posteriormente na aprendizagem supervisionada. Quando a quantidade de resultados bem sucedidos atingir um limiar predeterminado o agente sai do estado “Processing successful behaviours” e vai para o estado “Supervised learning and update knowledge base”, onde é realizado o aprendizado de novas regras reativas. Após gerar essas novas regras e atualizar a base de conhecimento, o agente volta ao estado “Monitoring new intrusions”. Assim, o agente pode ter múltiplos estados paralelos quando, por exemplo, surgem várias intrusões ao mesmo tempo. No entanto, na implementação atual do agente, ele trata apenas uma nova percepção por vez.

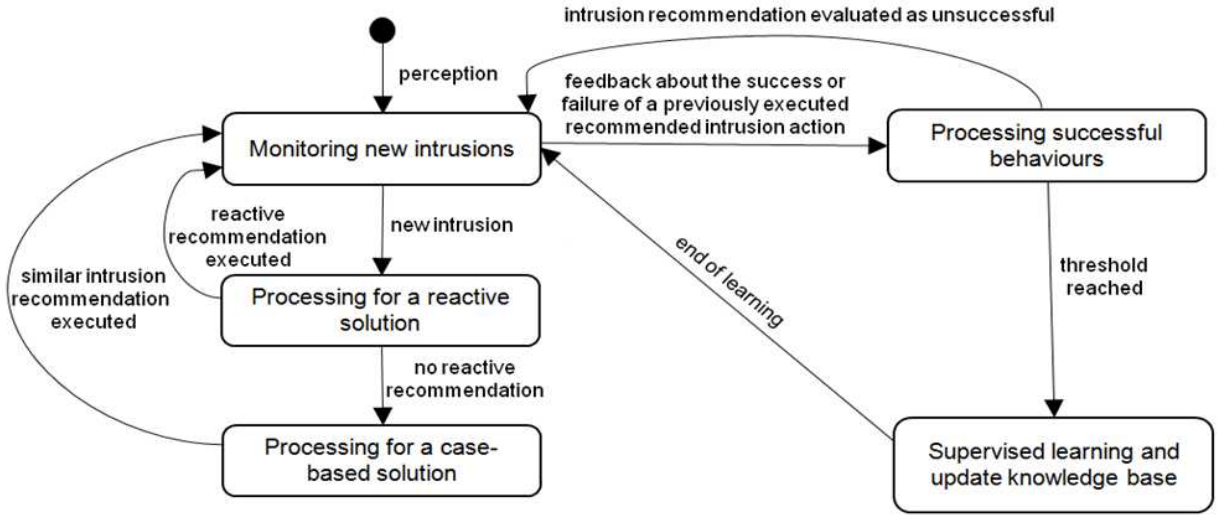


Figura 39. Diagrama de estados do agente RBC com aprendizagem

A Figura 40 ilustra o diagrama de atividades por que passa o agente RBC com aprendizagem. Após um agente ter uma nova percepção, o componente "Perception Interpretation" analisa se é uma percepção de realimentação ou uma nova percepção. Se for uma nova percepção, ela será tratada pelo componente reativo e se for uma percepção de realimentação será encaminhada para o componente de aprendizagem. Quando se trata de uma nova percepção, primeiro o componente reativo tenta encontrar uma ação que atenda a mesma, se encontrar, uma ação reativa é executada no ambiente. Se não encontrar, o componente deliberativo realiza um processo de inferência para encontrar uma ação para a percepção e, se for encontrada uma ação deliberativa, a mesma é executada no ambiente. Quando se trata de uma percepção de realimentação, o subcomponente crítico avalia se a mesma atendeu ao padrão de desempenho do agente. Se a ação foi bem-sucedida ela é armazenada para compor o conjunto de treinamento do agente. Quando o agente atinge uma determinada quantidade de exemplos de treinamento, a aprendizagem supervisionada é realizada e um classificador é gerado. Esse classificador é utilizado para compor as regras reativas da base de conhecimento do agente.

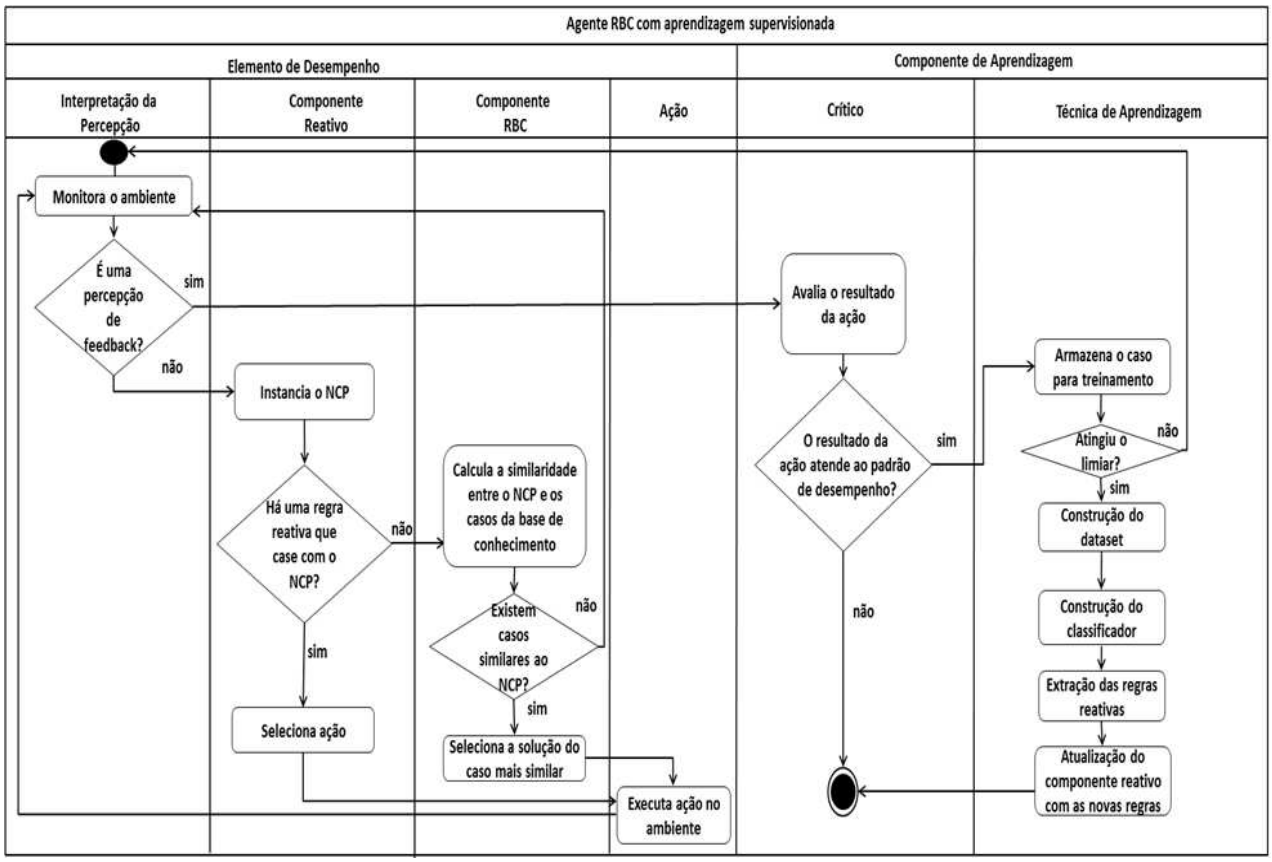


Figura 40. Diagrama de atividades do agente RBC com aprendizagem

3.2 Componentes da arquitetura HyLAA

O agente foi implementado na linguagem Prolog, utilizando a ferramenta para construção de agentes Chimera, integrada ao ambiente LPA Prolog [41]. A ONTOID, ontologia utilizada como base de conhecimento do agente desenvolvido, inclui um conjunto de casos para detecção de intrusão (que são as instâncias da ontologia), mas a mesma não possui regras reativas. As regras reativas foram obtidas através de aprendizagem supervisionada utilizando árvores de decisão.

3.2.1 ONTOID: A base de conhecimento do agente HyLAA

A Figura 41 ilustra parte da ONTOID representando intrusões de rede. A classe raiz da ontologia é "IntrusionCase" que possui os relacionamentos não-taxonômicos "hasProblem" e "hasSolution" com as classes "IntrusionCaseProblem" e "IntrusionCaseSolution", representando, respectivamente, o problema e a solução de um caso. A instanciação das sublcasses "IntrusionCaseProblem" e "IntrusionCaseSolution" varia conforme o domínio utilizado. Na ONTOID as propriedades do problema e da solução de uma intrusão a rede são representados pelas subclasses "IntrusionCaseProblem" e "IntrusionCaseSolution", respectivamente.

Um exemplo de um caso instanciado na ONTOID é ilustrado na Figura 41. Nela temos a instância de um caso, chamada "IntrusionCase_1" que possui o problema (instância "IntrusionProblem_1") contendo um conjunto de dados sobre uma conexão de rede que caracterizam um determinado tipo de intrusão e a solução a essa intrusão (instância "IntrusionSolution_1") contendo a identificação dessa intrusão e os procedimentos a serem realizados para evitar novas intrusões desse tipo.

O agente HyLAA quando em execução fica constantemente monitorando o ambiente. Assim que ele tem uma nova percepção, ela é tratada pelo componente "Perception Interpretation", e ele primeiro verifica se é uma percepção de realimentação, contendo resultado da execução de uma ação no ambiente. Se não for uma percepção de realimentação é um Novo Caso Problema (NCP). Se a percepção for um NCP, ela será tratada, preferencialmente, pelo componente "Reactive System", para sua execução imediata, e caso não existam regras reativas que satisfazem o NCP, ele será tratado pelo componente RBC. Quando o agente tem uma nova percepção, ele irá instanciar a ONTOID, conforme exemplificado na Figura 42.

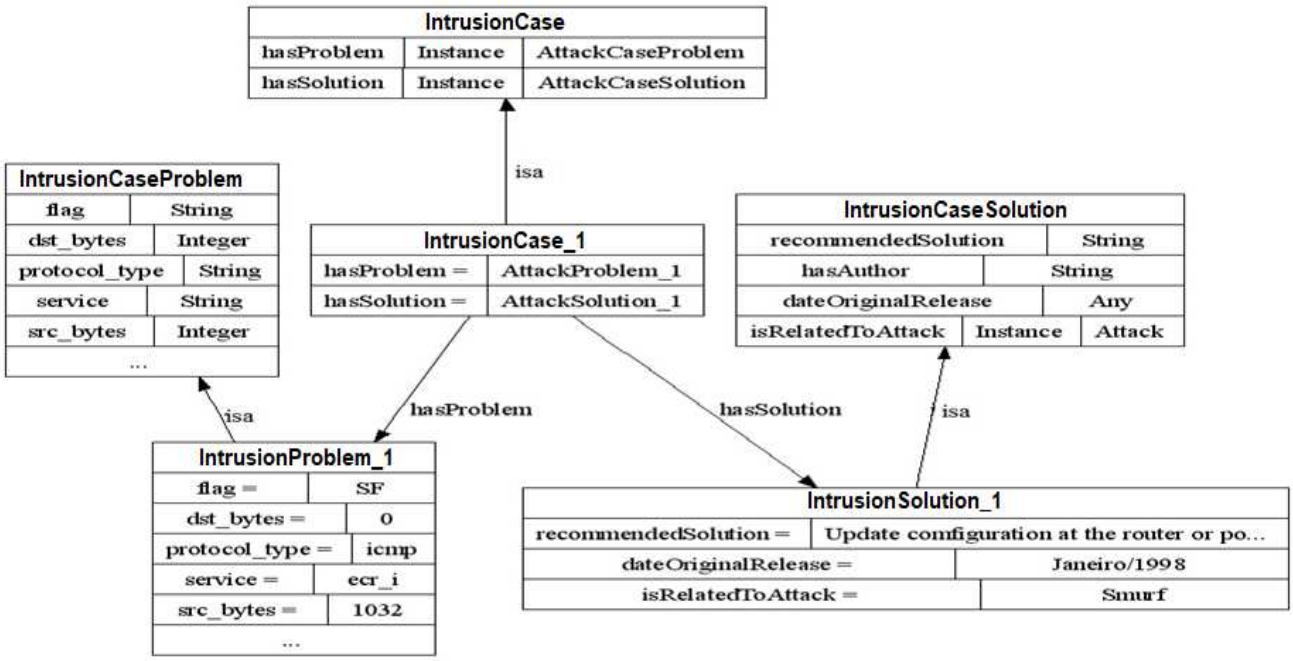


Figura 41. Exemplo de um caso instanciado na ONTOID

MonitoredPackage_1	
flag=	SF
count=	80
duration=	0
dst_bytes=	0
dst_host_count=	255
protocol_type=	icmp
service=	ecr_i
src_bytes=	520

Figura 42. NCP Instanciado “PackageMonitored_1”

Em seguida, o NCP instanciado é comparado com as regras reativas da base a fim de encontrar alguma solução reativa para o NCP. Por exemplo, ao comparar o NCP e a regra reativa ilustrada na Figura 43, há coincidência completa entre os valores dos atributos (protocol_type=icmp and service=ecr_i and flag=SF), assim a ação do agente será parte da ação da regra reativa (alert “Pod Intrusion”, “Configure individual hosts and routers to not respond to ICMP requests or broadcasts”). Na Figura 44 é mostrada uma tela com o resultado da execução da regra reativa anterior.

```
if protocol_type=icmp and service=ecr_i and flag=SF  then
   intrusion is pod
   alert “Pod Intrusion”.
“Configure individual hosts and routers to not respond to ICMP requests or broadcasts.”
```

Figura 43. Regra reativa para uma intrusão do tipo POD

```
|?-start_HyLAA_agent.
*CBR Hybrid Agent waiting new perceptions…*
# Abolishing f:\HyLAA_Agent\knowledge_base\ontoid.pl
# 0.669 seconds to consult f:\HyLAA_Agent\knowledge_base\ontoid.pl
New Case Problem: MonitoredPackage_1
Recommended solution (reactive): *Configure individual hosts and routers to not respond to ICMP requests or broadcasts
```

Figura 44. Ação recomendar solução para uma intrusão de Pod

Na Figura 45, temos outro exemplo de NCP instanciado na ONTOID (PackageMonitored_2), no entanto, nesse caso não há coincidência completa entre os valores dos seus atributos com nenhuma das regras reativas. Assim, esse NCP será tratado pelo componente RBC.

MonitoredPackage_2	
flag=	S0
count=	0
duration=	0
dst_bytes=	0
dst_host_count=	255
protocol_type=	ip
service=	ecr_i
src_bytes=	520

Figura 45. NCP Instanciado “PackageMonitored_2”

3.2.2 O componente deliberativo

Para desenvolver o componente deliberativo do agente utilizou-se o raciocínio baseado em casos. O componente RBC, é responsável por calcular a similaridade entre o NCP e os problemas já existentes na base de conhecimento. A Figura 46 mostra, com maiores detalhes, o subcomponente “CBR System”. Cada nova percepção do agente corresponde a um novo caso problema que é representada como uma instância da ontologia no subcomponente “Case representation”. Esse novo caso problema é então comparado com os casos da ontologia e para

cada um deles é calculada a similaridade através de um modelo de similaridade.

Um modelo de similaridade formaliza como os casos deverão ser comparados. Ele é usado pelo mecanismo "Retrieval and similarity analysis" para identificar os casos mais semelhantes ao novo caso de problema. Assim, o agente deve ser capaz de selecionar o caso com maior similaridade e reutilizar a sua solução. Antes da solução ser recomendada, pode ocorrer alguma adaptação. Isso é feito pelo componente "Adaptation" que contém um conjunto de regras de adaptação e conhecimento de domínio.

Finalmente, a solução obtida é executada no ambiente. Também pode haver aprendizagem, através do componente "Learning". O agente de software deve ser capaz de perceber os impactos desta solução no ambiente, por exemplo, se o problema foi resolvido com sucesso através da solução aplicada. Se for recorrentemente bem-sucedido, o mecanismo de aprendizagem irá adicionar o novo caso resolvido à base de conhecimento.

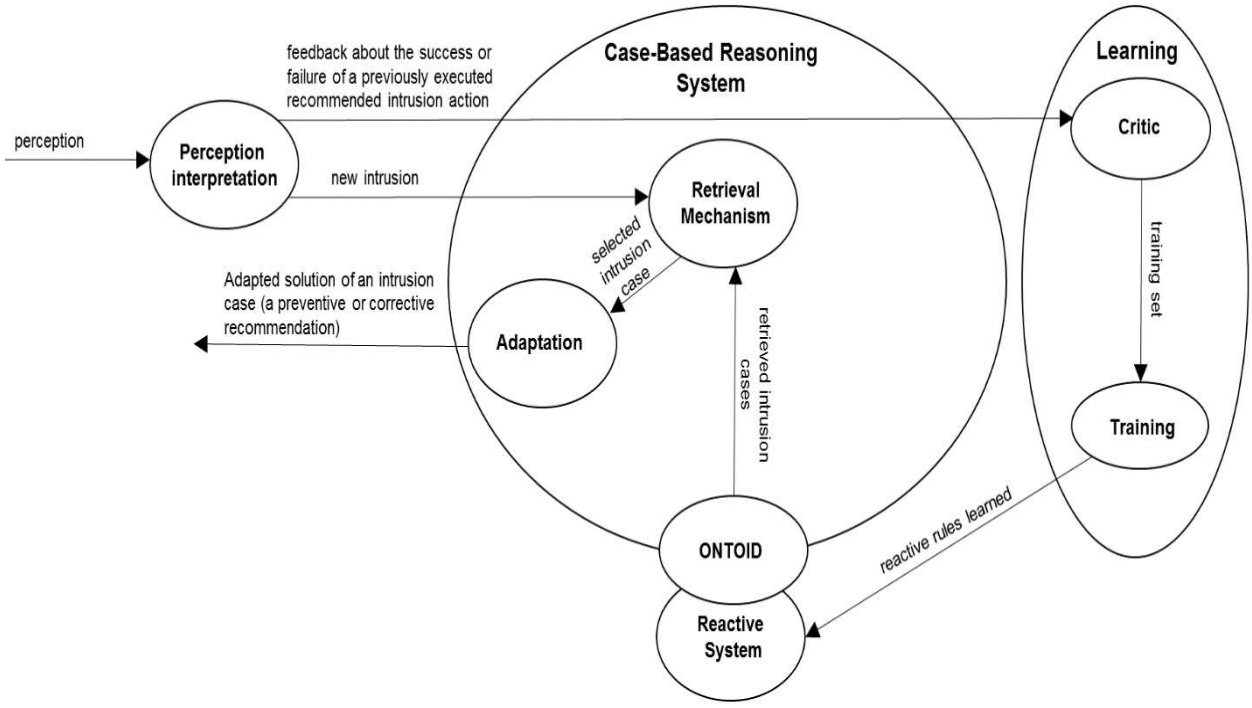


Figura 46. O Componente "CBR System"

Para calcular a similaridade entre um NCP e os casos da ONTOID, foram utilizadas as fórmulas (1) e (2):

$$\text{sim\_case}(NCP, KBCP) = \sum_{j=1}^{n} w_j \quad \text{sim\_attribute}_j(Cncp_j, Ckbcp_j) \qquad (1)$$

onde,

$$\sum_{j=1}^{n} w_j = 1$$

NCP – New Case Problem;

KBCP – Knowledge Base Case Problem;

j é o índice associado ao j-ésimo atributo do problema utilizado na recuperação;

w_j é o peso atribuído ao j-ésimo atributo;

$sim\_attribute_j$ é a função que calcula a similaridade entre os valores do j-ésimo atributo do NCP e do KBCP, segundo a fórmula a seguir:

$$sim\_attribute_j(Cncp_j, Ckbcp_j) = \frac{2 * |Cncp_j \cap Ckbcp_j|}{|Cncp_j| + |Ckbcp_j|} \quad (2)$$

$Cncp_j$ é o conjunto formado pelos valores dos atributos do NCP, e por sua hierarquia de superconceitos, quando houver, por exemplo: $Cncp_{duration} = 0$ onde, 0 é o valor do atributo "duration" do NCP;

$Ckbcp_j$ é o conjunto formado pelo(s) valor(s) do atributo do KBCP e por sua hierarquia de superconceitos, quando houver, por exemplo $Ckbcp_{service}$ = 'http' onde, 'http' é o valor do atributo "service" do KBCP.

Um exemplo do cálculo de similaridade, com parte dos atributos do problema, é mostrado na Tabela 3. Na primeira coluna estão as propriedades e valores de um NCP, na segunda coluna as propriedades e valores de um KBCP, a terceira coluna contém o valor de similaridade entre os atributos do NCP e de um KBCP, aplicando a fórmula 2 e considerando que todos os atributos tenham a mesma relevância, ou seja, o mesmo peso (wj = 0,33) e, finalmente, a quarta coluna contém a similaridade entre o NCP e um KBCP obtida aplicando-se a fórmula 1. O conjunto de todos os valores de propriedades representa um problema de intrusão. No entanto, alguns valores de propriedades são mais importantes para identificar um tipo particular de intrusão. Por exemplo, o tamanho do pacote IP é 65.535 bytes. A intrusão pod ("Ping da Morte") é caracterizada por envio de pacotes de "ping" maiores do que 65.535 bytes (pacotes mal formados). A propriedade "wrong_fragment" tem valor 0 quando um pacote está corretamente

formado e 1 se for mal formado. Assim, “wrong_frangment” é a propriedade mais relevante para identificar uma intrusão “pod”.

Tabela 3. Exemplo de cálculo de similaridade, utilizando parte dos atributos de um problema

NCP		**KBCP**		**Similaridade entre os valores dos atributos do NCP e do KBCP**	**Similaridade entre o NCP e o KBCP**
Propriedade	**Valor**	**Propriedade**	**Valor**		
count	487	Count	385	0	0,66
duration	0	duration	0	1	
flag	SF	flag	SF	1	

A Figura 47 mostra um exemplo de um problema similar ao NCP (Figura 45) recuperado da base de conhecimento com sua correspondente solução. Assim, a solução dada por esse caso a ser executada no ambiente será a recomendação “atualizar a configuração do roteador” (“Block directed broadcast traffic coming into network”). A Figura 48 ilustra a execução dessa ação no ambiente.

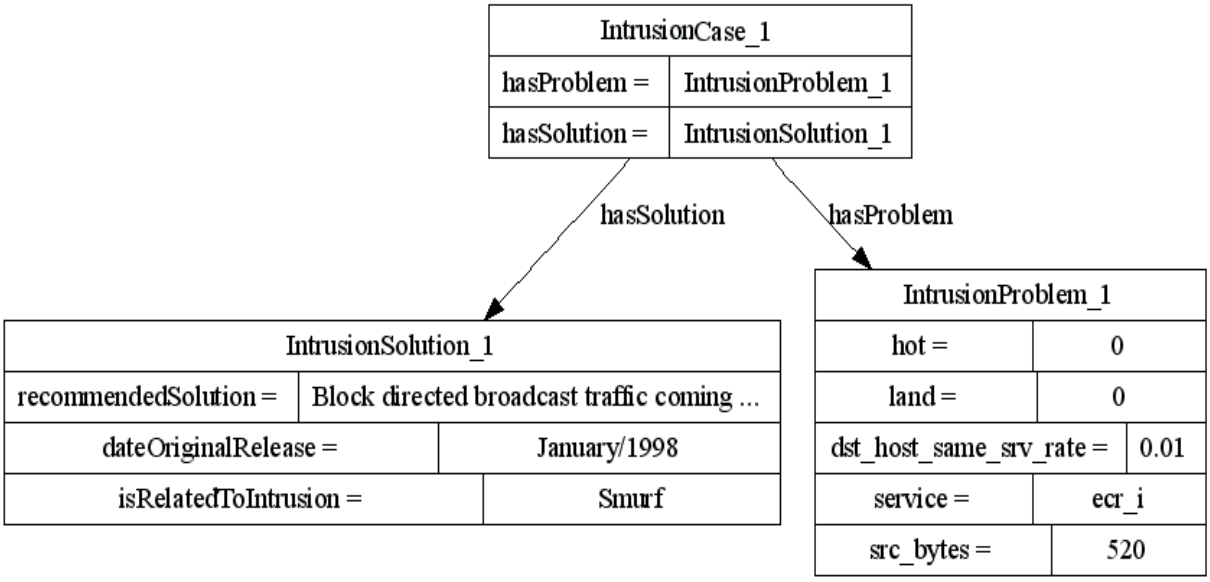


Figura 47. Caso similar ao NCP recuperado da ONTOID

```
|?-start_HyLAA_agent.
*CBR Hybrid Agent waiting new perceptions…*
# Abolishing f:\HyLAA_Agent\knowledge_base\ontoid.pl
# 0.969 seconds to consult f:\HyLAA_Agent\knowledge_base\ontoid.pl
New Case Problem: MonitoredPackage_2
Most similar problem: O1#IntrusionProblem_1
Intrusion type: Smurf
Similarity value: 0.90
Recommended solution:
 *Block directed broadcast traffic coming into network
 *Configure hosts and routers to not respond to ICMP echo requests
```

Figura 48. Ação recomendação de uma solução a intrusão “Smurf”

3.2.3 Sistema de aprendizagem do agente HyLAA

Uma percepção de realimentação é o resultado da execução de uma ação no ambiente percebido pelo agente. Ela é definida pela tripla <percepção, ação, resultado> denominado de percepção de realimentação (feedback). Esse tipo de percepção é tratada pelo componente de aprendizagem, mas precisamente pelo subcomponente Crítico. Nos estudos de casos deste trabalho, o resultado da execução de uma ação no ambiente foi definido explicitamente pelo usuário como “Successful” ou “Fail” baseado no conjunto de dados NSL-KDD que continha além da percepção correspondente a uma intrusão, a sua solução, ou seja, o tipo de intrusão associado. Desta forma, quando o agente acertava qual o tipo de intrusão de uma determinada percepção, era acrescentado ao conjunto <percpeção, ação> o resultado <Successful> e quando ele errava era acrescentado o resultado <“Fail”> a esse conjunto, como representado nos exemplos a seguir.

```
`13,0.00,0,255,0.01,0.20,0.00,0.93,0.00,237,0.00,0.22,0.00,2088,RSTR,0,0,0,0,1,0,0,0,0,0,0,0,tcp,0.92,0,1.00,0.08,http,72564,13,0.00,0.92,0.08,0,0,0,Neptune# ,Sucessful`. => Recomendação correta
`13,0.00,0,255,0.01,0.20,0.00,0.93,0.00,237,0.00,0.22,0.00,2088,RSTR,0,0,0,0,1,0,0,0,0,0,0,0,tcp,0.92,0,1.00,0.08,http,72564,13,0.00,0.92,0.08,0,0,0, PoD#, Fail`. = Recomendação incorreta
```

Como se utilizou uma forma explícita, informada pelo usuário, para se obter o resultado da ação do ambiente o padrão de desempenho (regras utilizadas pelo Crítico) ficaram bastante simplificadas como ilustradas no algoritmo abaixo (Figura 49):

```
if (percepcao = X) and (acao = Y) and (resultadoacao= “Sucessful”) then
store example(percepcao, acao)
```

Figura 49. Exemplo de padrão de desempenho

Assim, dos exemplos anteriores (Neptune e Pod), apenas o exemplo abaixo seria armazenado para treinamento, como ilustrado abaixo:

```
`13,0.00,0,255,0.01,0.20,0.00,0.93,0.00,237,0.00,0.22,0.00,2088,RSTR,0,0,0,0,1,0,0,0,0,0,0,0,tc
p,0.92,0,1.00,0.08,http,72564,13,0.00,0.92,0.08,0,0,0, Neptune# `.
```

O agente continua armazenando os casos, cujas ações tiveram boa avaliação pelo subcomponente “Critic” (resultado = “Sucessful”), até que o número de elementos do conjunto de treinamento com exemplos positivos atinja um determinado limiar, considerado suficiente para o aprendizado. Quando o conjunto de dados de treinamento atinge esse limiar, o subcomponente “Critic” o envia para o subcomponente “Training”. Em seguida, o subcomponente “Training” gera um classificador composto de um conjunto de regras reativas. A Figura 50 mostra o exemplo de um conjunto de regras reativas aprendidas. Por fim, o subcomponente de treinamento atualiza o componente “reactive system” do agente com essas novas regras aprendidas.

```
if intrusion_category=DoS and protocol_type=tcp    and service=http then intrusion is land
if intrusion_category=DoS and protocol_type=ftp    and service=http then intrusion is land
if intrusion_category=DoS and protocol_type=smtp   and service=http then intrusion is land
if intrusion_category=DoS and protocol_type=telnet and service=http then intrusion is land
if intrusion_category=DoS and protocol_type=icmp   and src_bytes<=1032 then intrusion is smurf
if intrusion_category=DoS and protocol_type=telnet and src_bytes>1032 then intrusion is pod
```

Figura 50. Exemplo de um conjunto de regras reativas aprendidas

3.3 Síntese

Este capítulo apresentou a arquitetura híbrida HyLAA. Primeiramente, foi apresentada uma visão estática da arquitetura contendo uma visão geral da arquitetura e a descrição de cada um dos componentes individualmente e, logo após, uma visão dinâmica do funcionamento da arquitetura, mostrando o seu funcionamento completo, ilustrado através dos diagramas de atividade e estado. Em seguida, apresentou-se a aplicação da arquitetura no desenvolvimento de um agente do agente HyLAA cujo objetivo é a detecção de intrusões a redes de computadores utilizando aprendizagem supervisionada.

Uma das vantagens da utilização da arquitetura HyLAA é a aprendizagem de comportamento deliberativo frequente, tornando-o reativo. Ao aprender regras reativas de forma automática, diminuem-se os custos de construção das bases de conhecimento dos agentes desenvolvidos. Outra vantagem da arquitetura HyLAA é que a mesma é baseada em ontologias, isto é, a base de conhecimento do agente deve ser representada em ontologias. As ontologias são ideais para a construção de bases conhecimento por vantagens, como maior expressividade semântica, descrição formal do conhecimento, reusabilidade, facilidade de adaptação e integração.

No próximo capítulo, a efetividade e o desempenho do agente HyLAA serão avaliados. A efetividade será avaliada através do desenvolvimento de um estudo de caso no qual o agente terá que detectar um conjunto de novas intrusões. A efetividade do agente é avaliada em termo das medidas de “precision” e “recall”, essas medidas avaliam capacidade de realizar recomendações relevantes e de ocultar recomendações irrelevantes. O desempenho do agente é avaliado medindo o tempo gasto para detectar e recomendar uma solução a cada nova intrusão.

4. AVALIAÇÃO

Este capítulo apresenta uma avaliação da arquitetura de HyLAA. A avaliação foi conduzida através do desenvolvimento de quatro estudos de casos, cujo objetivo é demonstrar a hipótese de pesquisa, definida na seção 1.3, ou seja, mostrar que os agentes híbridos desenvolvidos com a arquitetura HyLARA são mais efetivos que um agente reativo ou deliberativo agindo isoladamente ou que um agente híbrido que combina os dois comportamentos, mas sem aprendizagem. A Tabela 4 mostra a organização dos quatro estudos de casos, incluindo o objetivo de cada um deles, o método empregado na sua realização e os resultados esperados. Nos apêndices 1, 2 e 3 deste manuscrito está disponível a documentação da implementação do agente, da sua execução e exemplos de reprodução das experiências realizadas neste capítulo.

O primeiro estudo de caso avalia a efetividade do componente RBC do agente quando executado isoladamente. Esse componente corresponde a um agente com capacidade de raciocínio analógico sem aprendizagem. O estudo de caso foi dividido em duas experiências: a primeira avalia a efetividade do agente na recomendação de soluções perante intrusões considerando uma distribuição uniforme dos pesos dos atributos considerados no cálculo de similaridade realizado no raciocínio por analogia enquanto a segunda experiência avalia o impacto na efetividade ao se utilizar pesos maiores para os atributos mais relevantes.

O segundo estudo de caso avalia a acurácia do componente de aprendizagem do agente isoladamente, isto é, a capacidade que o classificador aprendido pelo componente de aprendizagem tem de classificar corretamente uma dada intrusão. O classificador aprendido pelo componente de aprendizagem que consiste de um conjunto de regras reativas é incorporado à ONTOID, base de conhecimento do agente HyLAA, atualizando assim o seu comportamento reativo. Este estudo de caso foi organizado em três experiências, cada uma utilizando um conjunto de treinamento diferente para aprendizagem do classificador. O objetivo destas experiências é analisar o impacto do conjunto de treinamento na acurácia do classificador.

O terceiro estudo de caso avalia a efetividade e o desempenho do agente híbrido na medida em que ele aprende, isto é, cada vez que o agente aprende um novo conjunto de regras a sua efetividade é avaliada. Este estudo de caso foi dividido em duas experiências. A primeira avalia a variação na efetividade do agente na medida em que ele aprende novo comportamento reativo. A segunda experiência avalia os ganhos na efetividade do agente após usar o último conjunto de regras reativas aprendidas para detectar o mesmo conjunto de intrusões da experiência 1.

O quarto estudo de caso avalia a efetividade e o desempenho do agente híbrido sem aprendizagem, isto é, apenas combinando os comportamentos RBC e reativo. O agente híbrido combina o componente reativo usando um conjunto de regras reativas pré-definidas e o componente RBC para detecção de intrusões. Para este estudo de caso foram desenvolvidas duas experiências. A primeira experiência usa o mesmo conjunto de 50 percepções do estudo de caso 1. O objetivo é comparar a efetividade do agente híbrido sem aprendizagem com a execução apenas do componente RBC. A segunda experiência usa um conjunto de 500 percepções, as mesmas usadas no quarto estudo de caso. O objetivo é comparar a efetividade e o desempenho do agente híbrido sem aprendizagem com o agente híbrido com aprendizagem.

Finalmente os resultados obtidos nos diferentes estudos de casos são comparativamente analisados mostrando os ganhos em termos de efetividade e desempenho do agente híbrido com e sem aprendizado (terceiro e quarto estudo de caso, respectivamente) e também os ganhos em termos de efetividade, desempenho, esforço de desenvolvimento e adaptabilidade do agente híbrido com aprendizagem perante o agente híbrido sem aprendizagem.

Tabela 4. Organização dos Estudos de Casos

	Objetivo	**Método**	**Resultados Esperados**
Estudo de Caso 1	Avaliar a efetividade do componente RBC em termo das medidas "precision" e "recall"	Exp. 1: Utilizar diferentes pontos de corte (valores mínimos de similaridade). Exp.2: Utilizar pesos maiores para os atributos mais relevantes. (seção 4.1)	Boa efetividade do componente RBC, ou seja, capacidade de recomendar soluções relevantes e ocultar soluções irrelevantes
Estudo de Caso 2	Avaliar a acurácia do classificador aprendido	Exp. 1: Utilizar um conjunto de treinamento com 150 exemplos. Exp. 2 : Utilizar um conjunto de treinamento com 500 exemplos. Exp. 3: Utilizar um conjunto de treinamento com 1000 exemplos. (seção 4.2)	Boa capacidade para classificar corretamente uma intrusão (alta acurácia do classificador aprendido)
Estudo de Caso 3	Avaliar a efetividade e desempenho do agente híbrido com aprendizagem	Executar o agente híbrido e avaliar a sua efetividade na medida em que ele aprende, isto é, a cada novo conjunto de regras reativas aprendidas. (seção 4.3)	Maior efetividade do comportamento híbrido com aprendizagem do que o RBC isoladamente Melhor efetividade na medida em que o agente aprende
Estudo de Caso 4	Avaliar a efetividade e desempenho do agente híbrido sem aprendizagem	Execução do agente híbrido sem aprendizagem e comparar os resultados com o resultado do estudo de caso 3. (seção 4.4)	Maior efetividade e desempenho do comportamento híbrido sem aprendizagem do que o RBC ou reativo isoladamente

4.1 Estudo de caso 1: avaliação do componente RBC

Esta seção descreve o estudo de caso realizado para avaliar a efetividade do componente RBC. O objetivo deste estudo de caso é avaliar a capacidade do componente RBC de recomendar soluções relevantes para uma dada intrusão e ocultar possíveis soluções irrelevantes através, respectivamente, da medição dos valores de "recall" e "precision" para um conjunto de intrusões. Para isso, o componente RBC do agente executa as seguintes ações no ambiente, conforme ilustrado na Figura 37: "Monitorar o ambiente de rede", onde o agente monitora a rede à espera da chegada de novos pacotes de dados referentes a uma sessão de conexão; "Detectar intrusões", na qual o agente analisa se os pacotes de dados percebidos na ação anterior se referem a uma intrusão ou a uma conexão normal e "Recomendar Soluções", na qual o agente sugere ao administrador da rede uma solução à intrusão identificada na ação anteriormente executada. A Tabela 5 resume as duas experiências realizadas, incluindo o seu objetivo, a metodologia empregada e os resultados esperados.

Tabela 5. Organização do Estudo de Caso 1

	Objetivos	**Metodologia**	**Resultados Esperados**
Exp. 1	Avaliar a efetividade do componente RBC considerando diferentes valores mínimos de similaridade como pontos de corte	Utilizar diferentes pontos de corte para calcular a similaridade Avaliar a efetividade através das medidas de "precision" e "recall" Analisar os resultados	Boa efetividadade para determinados pontos de corte
Exp. 2	Analisar o impacto na efetividade das recomendações ao atribuir pesos maiores aos atributos mais relevantes	Identificar os atributos mais relevantes e atribuir pesos maiores a eles Avaliar os resultados Comparar os resultados com a experiência 1	Boa efetividade utilizando pesos maiores para os atributos mais relevantes

A apresentação deste estudo de caso está organizada em três seções. A seção 4.1.1 descreve o método utilizado para organizar e realizar as experiências de avaliação deste estudo de caso. A seção 4.1.2 apresenta os resultados obtidos em cada uma das experiências realizadas. Finalmente, a seção 4.1.3 analisa os resultados obtidos de acordo aos objetivos de avaliação.

4.1.1 Método

O agente foi implementado na linguagem Prolog, utilizando a ferramenta para construção

de agentes Chimera, integrada ao ambiente LPA Prolog [41]. A base de conhecimento do agente é a ontologia ONTOID, descrita na seção 3.1. A ONTOID foi implementada originalmente em OWL, utilizando o editor de ontologias Protégé, e posteriormente mapeada para a linguagem Prolog utilizando a ferramenta Thea [72]. Para o povoamento da ontologia foi utilizado o conjunto de dados NSL-KDD [53] [57]. O NSL-KDD é um conjunto de dados de simulação de intrusões a uma rede de computadores que inclui os dados de uma sessão de conexão a uma determinada rede e a correspondente classificação desses dados em um determinado tipo de intrusão.

A efetividade das soluções recomendadas pelo agente foi avaliada em termos das tradicionais medidas de avaliação da área da recuperação de informação, "recall" e "precision" [3]. O "recall" é a razão entre o número de soluções relevantes recuperadas e o número total de soluções relevantes da base de conhecimento e "precision" é a razão entre o número de soluções relevantes recuperadas e o número total de casos recuperados. Para analisar os resultados, o gráfico de "precision-recall", que mostra a evolução dos valores de "precision" em função dos valores de "recall" foi construído. Neste gráfico, os valores de "recall" são listados no eixo X e os de "precision" no eixo Y. Uma boa efetividade é atingida quando a "precision" não é comprometida para altos valores de "recall", isto é, mesmo que o agente aumente a quantidade de soluções recomendadas ao usuário elas continuam a ser relevantes em sua maioria. A "precision" diminui na medida em que o "recall" aumenta e vice-versa [21]. A primeira experiência foi desenvolvida visando determinar o ponto de corte ideal para a obtenção de uma boa efetividade através de um balanço nos valores de "recall" e "precision", de forma que altos valores de "recall" não comprometam a "precision" do sistema. O ponto de corte define o valor mínimo de similaridade necessário para que ao menos uma solução seja recuperada pelo agente. É necessário identificar o ponto de corte ideal, pois ao utilizar pontos de corte com valores muito altos, o agente pode não recuperar nenhuma solução para uma dada intrusão, enquanto que pontos de corte muito baixos podem fazer com que o agente recomende soluções com valores de similaridade muito baixos e, portanto, irrelevantes. Desse modo, essa primeira experiência visa encontrar o ponto de corte ideal, isto é, o maior ponto de corte possível que retorne soluções com altos valores de "precision" para a maioria das intrusões identificadas. Os pontos de corte utilizados nessa experiência foram 0.7, 0.8 e 0.9. Esses valores foram escolhidos em testes realizados previamente, nos quais foram obtidas muitas soluções irrelevantes com pontos de cortes inferiores a 0.7 e muitas soluções relevantes não foram recuperadas com pontos de corte superiores a 0.9.

Na segunda experiência, os mesmos pontos de corte da experiência anterior foram utilizados, mas considerando pesos maiores para os atributos mais relevantes no cálculo de similaridade, conforme a medida especificada na seção 3.4, pois as intrusões a uma rede de computadores têm algumas características que as distinguem entre si. Assim, com esta segunda experiência, espera-se descobrir qual a importância dos atributos na identificação de uma intrusão, atribuindo-se pesos maiores aos atributos considerados mais relevantes e verificar se o resultado é superior a atribuição de pesos iguais. Nessa experiência os atributos mais relevantes foram selecionados de acordo com o trabalho de Olusola et al. [53].

Para simular as pecepções do agente nas duas experiências, foi utilizado um mesmo conjunto de 50 instâncias extraídas aleatoriamente do conjunto de dados NSL-KDD e 11.800 instâncias para povoar a base de conhecimento representando 37 tipos de intrusões diferentes. Cada instância possui um conjunto de 41 valores de atributos que caracterizam uma intrusão. Os atributos contém informações sobre uma sessão de conexão como o protocolo de comunicação utilizado, a quantidade de bytes transferidos, o serviço utilizado, etc. As 50 percepções correspondem a 10 tipos de intrusões diferentes (“neptune”, “guess_password”, “warezmaster”, “satan”, “smurf”, “back”, “portsweep”, “ipsweep”, “nmap”, “pod”), sendo cinco percepções para cada tipo de intrusão. Estas intrusões foram selecionados de acordo com a quantidade de instâncias disponíveis na base de conhecimento (as intrusões com mais instâncias foram selecionados prioritariamente) e a informação disponível na literatura sobre as características que diferenciam cada intrusões, ou seja, os atributos mais relevantes. A Tabela 6 mostra a quantidade de instâncias disponíveis na ONTOID para cada um dos 10 tipos de intrusões utilizadas como percepções.

Tabela 6. Conjunto de instâncias da ONTOID, classificadas por tipo de intrusão

Nome do tipo de intrusão	**Quantidade de instâncias na ONTOID**
neptune	1579
guess_passwd	1226
warezmaster	944
satan	727
smurf	627
back	359
portsweep	156
ipsweep	141
nmap	73
pod	41
Demais intrusões (land, worm, imap,etc)	5931

A seguir é apresentada uma breve descrição dos dez tipos de intrusões utilizados no conjunto das percepções do agente RBC [16], utilizando o dataset disponível em [52]:

- **“neptune”:** é um tipo de intrusão de negação de serviço, também conhecido como “syn flood”, no qual o atacante envia uma sequência de requisições ao servidor, causando sobrecarga no mesmo;
- **“guess_passwd”:** é uma intrusão que explora uma vulnerabilidade que ocorre quando os sistemas operacionais estão configurados com uma conta com perfil de convidado (um perfil de usuário normalmente com permissões bem limitadas) ativada e sem senha ou com uma senha padrão;
- **“warezmaster”:** é uma intrusão que explora uma falha dos servidores de FTP que estão configurados com permissão de escrita para o usuário com perfil de convidado. Ao encontrar esse tipo de falha, o atacante normalmente faz upload de arquivos ilegais para o servidor atacado e os compartilha, consumindo os seus recursos de armazenamento e de conexão à Internet;
- **“satan”:** essa intrusão se caracteriza pelos atacantes utilizarem ferramentas chamadas de scanners que varrem o sistema alvo tentando encontrar falhas de segurança. O que diferencia o “satan” dos demais tipos de intrusão que utilizam ferramentas de scanner é o seu padrão de conexão (tempo e os serviços que são escaneados);
- **“smurf”:** é um tipo de intrusão de negação de serviço no qual o atacante envia diretamente e através de terceiros, uma grande quantidade de requisições ICMP para o servidor alvo;
- **“back”:** é um tipo de intrusão de negação de serviço onde o atacante envia uma requisição a um servidor web com uma grande quantidade de barras na URL e o servidor fica indisponível porque não consegue processar esse tipo de requisição. Essa intrusão é detectada através da definição de um limiar normal e anormal, por exemplo, a partir de 100 barras na URL de uma requisição poderia ser considerada intrusão;
- **“portsweep”:** essa intrusão tenta identificar as portas abertas de um host através de ferramentas de scanner para atacá-lo;
- **“ipsweep”:** consiste em identificar os hosts de uma rede para atacá-lo, um método simples de fazer isso é através do envio de requisições “ping” a todos os hosts de uma rede e verificar quais respondem;

- **“nmap”:** é uma intrusão, assim como o “satan”, que utiliza scanners para varrer o sistema alvo detectando falhas de segurança para explorá-las. Essa intrusão pode ser detectado através do padrão de conexão (tempo, quantidade de portas escaneadas por vez e os tipos de serviços);
- **“pod”:** é uma intrusão de negação de serviço no qual o atacante utiliza o comando “ping” para enviar requisições com o objetivo de sobrecarregar o sistema alvo.

A Tabela 7 apresenta os atributos mais relevantes para cada um dos 10 tipos de intrusões utilizados nas percepções do agente, o peso atribuído a esses atributos e aos demais atributos. Os pesos foram definidos de forma que o valor atribuído aos atributos mais relevantes for sempre superior àqueles dos demais atributos.

Tabela 7. Conjunto dos atributos mais relevantes e seus pesos

Nome da Intrusão	**Atributos Relevantes**	**Peso**	
		Atributos Relevantes	**Demais Atributos**
neptune	count,diff_srv_rate,dst_host_count, dst_host_same_src_port_rate, dst_host_same_srv_rate, dst_host_serror_rate, dst_host_srv_serror_rate, flag, same_srv_rate, src_bytes, srv_diff_host_rate, srv_serror_rate	0,0321	0,0203
guess_passwd	service,flag,dst_bytes, num_failed_logins	0,0370	0,0237
warezmaster	dst_bytes, duration	0,0350	0,0238
satan	diff_srv_rate, rerror_rate, service	0,0333	0,0236
smurf	diff_srv_rate, dst_bytes, st_host_count, dst_host_same_src_port_rate, dst_host_srv_serror_rate, logged_in, protocol_type,same_srv_rate ,serror_rate,service,src_bytes	0,0333	0,0206
back	dst_bytes, src_bytes	0,0350	0,0238
portsweep	srv_rerror_rate	0,0300	0,0242
ipsweep	dst_host_same_src_port_rate		
nmap	src_bytes		
pod	wrong_fragment		
Demais intrusões		**0,02439**	

4.1.2 Resultados

A Tabela 8 apresenta a efetividade do componente RBC em termo das medidas de “precision” e “recall”. Observa-se que em ambas as experiências, na utilização do ponto de corte

0.7 obteve-se valores de “precision” muito baixos (uma média de 21% e 24% na primeira e segunda experiência, respectivamente). Isso significa que o sistema recuperou muitas soluções irrelevantes, enquanto que os valores de “recall” foram bons (aproximadamente 78% e 76% na primeira e segunda experiência, respectivamente). Com o ponto de corte 0.9 a “precision” teve o melhor resultado entre os três pontos de corte utilizados (93% e 94% na primeira e segunda experiência, respectivamente), porém o “recall” foi o mais baixo (29% e 30% na primeira e segunda experiência, respectivamente), assim o sistema deixou de recuperar muitos casos da base de conhecimento considerados relevantes. Já com o ponto de corte 0.8 pode-se observar que tanto a “precision” (61% e 63% na primeira e segunda experiência, respectivamente) quanto o “recall” (63% e 60% na primeira e segunda experiência, respectivamente) foram mais equilibrados. Na segunda experiência obteve-se valores maiores de *“precision”* em todos os pontos de corte utilizados em relação aos valores obtidos na primeira experiência e maiores valores de *“recall”* em 2 dos 3 pontos de corte utilizados, como pode ser observado na Tabela 8. Entretanto, a diferença entre esses valores foi pequena. Um dos fatores que contribuiram para isso foi o fato de que muitas das intrusões do NSL-KDD possuem muitas características em comum, em alguns casos, apenas um atributo diferencia uma intrusão de outra, então, mesmo aumentando os pesos dos seus atributos o resultado do cálculo de similariade continua muito próximo. A Figura 51 apresenta o gráfico “recall-precision” das duas experiências para o ponto de corte 0,9.

Tabela 8. Efetividade do componente RBC segundo os resultados obtidos nas experiências 1 e 2

Exp. 1 (Distribuição uniforme de pesos)			**Exp. 2 (Maiores pesos para os atributos mais relevantes)**		
Ponto de corte	Média *“precision”*	Média *“recall”*	Ponto de corte	Média *“precision”*	Média *“recall”*
0,7	21%	78%	0,7	24%	76%
0,8	61%	63%	0,8	63%	60%
0,9	93%	29%	0,9	94%	30%

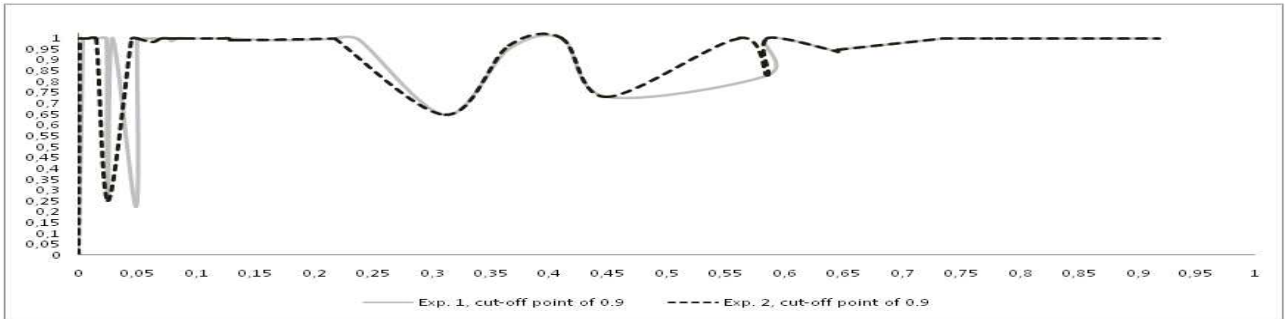


Figura 51. Gráfico de “recall-precision” para o ponto de corte 0.9

4.1.3 Discussão

No problema de recomendar soluções a intrusões em redes de computadores é importante garantir altos valores de “precision”, independentemente dos valores de “recall”, uma vez que é mais vantajoso recomendar poucas, mas boas soluções ao usuário, evitando soluções de tentativa-erro que são mais custosas tanto no esforço quanto no tempo gasto, fatores críticos da área de segurança da informação.

Apesar dos valores obtidos nas duas experiências serem muito próximos como apresentado na Tabela 8, tanto os resultados de “precision” quanto de “recall” obtidos na segunda experiência foram superiores à primeira. No entanto, em ambas as experiências, conforme o gráfico da Figura 51, houve queda brusca da “precision” em alguns pontos. Isso ocorre quando a quantidade de casos relevantes de um determinada intrusão armazenados na base de conhecimento é reduzida. Por exemplo, isso acontece nas intrusões “pod”, “ipsweep” e “portsweep”, que possuem somente 41, 141 e 156 instâncias na base de conhecimento, respectivamente, como pode ser verificado na Tabela 6.

Considerando os resultados obtidos nas duas experiências realizadas, conclui-se que o agente HyLAA realiza recomendações de soluções com boa efetividade utilizando o ponto de corte 0.9, porque, como apresentado na seção 4.1.2, mesmo quando os valores de “recall” aumentam, a “precision” se mantém alta, ou seja, mesmo quando a quantidade de soluções recomendadas aumenta, elas continuam sendo soluções relevantes.Também, percebe-se que a atribuição de pesos maiores aos atributos mais relevantes não representou uma melhora significativa nos valores de “precision” e “recall”. Por exemplo, conforme pode ser visualizado no gráfico da Figura 51, ao usar o ponto de corte 0.9, os resultados da experiência 2 foram superiores aos da experiência 1 em apenas 1% de precision e 1% de recall. Considerando que o esforço para definir os atributos mais relevantes e os seus respectivos pesos pode ser elevado de acordo com os possíveis tipos de intrusões que um IDS terá que lidar, deve ser analisado se um pequeno ganho de efetividade justifica o esforço na discriminação dos pesos dos atributos.

4.2 Estudo de caso 2: avaliação do componente de aprendizagem

Este segundo estudo de caso tem por objetivo avaliar a acurácia do componente de aprendizagem. Espera-se que as regras aprendidas, ou seja, o classificador gerado, apresente alta acurácia, isto é, alta taxa de classificação correta de exemplos, pois as mesmas terão impacto direto na efetividade global do agente híbrido desenvolvido, uma vez que o mesmo utiliza o comportamento reativo como prioritário. Isto significa que o agente híbrido sempre tentará detectar uma nova intrusão de forma reativa, devido ao maior desempenho deste tipo de comportamento e apenas no caso de não haver regras reativas, o mesmo agirá de forma deliberativa. Para realizar o estudo de caso foram realizadas três experiências, uma para cada conjunto de treinamento utilizando, respectivamente 150, 500 e 1000 exemplos, conforme descrito na Tabela 9 e um conjunto de 50 percepções para teste. A realização de três experiências tem por objetivo analisar a relação entre o tamanho do conjunto treinamento com a acurácia do classificador.

Tabela 9. Organização do Estudo de Caso 2

	Objetivos	**Método**	**Resultados Esperados**
Experiência 1	Avaliar a acurácia do classificador com um conjunto de 150 exemplos de treinamento	150 exemplos para treinamento e 50 exemplos para teste Avaliar a acurácia do classificador Analisar os resultados	Boa acurácia do classificador
Experiência 2	Avaliar a acurácia do classificador com um conjunto de 500 exemplos de treinamento	500 exemplos para treinamento e 50 exemplos para teste Avaliar a acurácia do classificador Analisar os resultados	
Experiência 3	Avaliar a acurácia do classificador com um conjunto de 1000 exemplos de treinamento	Utilizar 1000 exemplos para treinamento e 50 exemplos para teste Avaliar a acurácia do classificador Analisar e comparar os resultados	

Na subseção 4.2.1, o método de avaliação utilizado para esse novo estudo de caso é descrito, na subseção 4.2.2 os resultados obtidos nas experiências realizadas são apresentados e, por fim, na subseção 4.2.3, os resultados obtidos são analisados.

4.2.1 Método

Para realizar as experiências desta avaliação, adotou-se o método Holdout [32] no qual o conjunto de exemplos disponíveis é dividido em duas partes: o conjunto de treinamento e o conjunto de teste. O conjunto de treinamento é uma coleção de exemplos utilizada para a construção do classificador e o conjunto de teste é utilizado para avaliação da acurácia desse

classificador. Segundo Sokolova e Lapalme [65], a acurácia permite avaliar a efetividade global de um classificador e leva em consideração o número de exemplos que correspondem a uma intrusão corretamente classificados em um dado tipo de intrusão ("true positive"); o número de exemplos que não correspondem a uma intrusão (trata-se de uma sessão de conexão normal) corretamente classificados ("true negative"); o número de exemplos que correspondem a uma intrusão incorretamente classificados ("false negative"); e o número de exemplos que não correspondem a uma intrusão e foram classificados incorretamente em um dado tipo de intrusão ("false positive"). Assim, a acurácia é calculada através da seguinte fórmula:

$$\text{Acurácia} = \frac{\text{tp}+\text{tn}}{\text{tp}+\text{fn}+\text{fp}+\text{tn}} \quad (3)$$

onde,

tp ("true positive") é o número de exemplos corretamente classificados em um tipo dado de intrusão;
tn ("true negative") é o número de exemplos corretamente classificados como não intrusão;
fp ("false positive") é o número de exemplos classificados incorretamente em um dado tipo de intrusão;
fn ("false negative"), número de exemplos classificados incorretamente como não intrusão.

A Tabela 10 mostra como as três experiências estão organizadas. Para a construção do classificador são utilizados três conjuntos de treinamento diferentes contendo 150, 500 e 1000 exemplos, respectivamente. Para o teste, um conjunto de 50 percepções (as mesmas do estudo de caso 1, para fins de comparação) é utilizado. A técnica de aprendizagem adotada é a aprendizagem supervisionada utilizando árvores de decisão e o algoritmo C4.5 [55][59].

Tabela 10. Metodologia de avaliação do classificador aprendido

Experiências	**Percepção (Conjunto de Teste)**	**Conjunto de Treinamento**	**Técnica de Aprendizagem**	**Algoritmo de Aprendizagem**
Experiência 1	50 exemplos (obtidas do NSL-KDD, as mesmas do estudo de caso 1)	150 exemplos de treinamento (obtidas aleatoriamente do NSL-KDD)	Aprendizagem supervisionada utilizando árvores de decisão	C4.5 [55][59]
Experiência 2		500 exemplos de treinamento (obtidas aleatoriamente do NSL-KDD)		
Experiência 3		1000 exemplos de treinamento (obtidas aleatoriamente do NSL-KDD)		

O algoritmo utilizado

A técnica utilizada para a aprendizagem do agente desenvolvido é o aprendizado indutivo, isto é, através de um conjunto de exemplos são obtidas regras genéricas. Os exemplos utilizados são rotulados e funcionam como um "professor" para o algoritmo de aprendizagem, informando a classe que corresponde a cada exemplo.

Na Figura 52, o exemplo da primeira linha da tabela <13, tcp, telnet, SF, 118 > corresponde a uma intrusão do tipo "guess_password". Cada linha contém exemplo de um determinado tipo de intrusão. O conjunto de todos os exemplos são usados pelo algoritmo de aprendizagem para aprender as regras de classificação.

Para realizar a aprendizagem de novas regras reativas foi adotado o algoritmo C4.5, desenvolvido por Ross Quinlan [55] que permite a construção de árvores de decisão. A versão utilizada pelo agente HyLAA é uma implementação desse algoritmo na linguagem Java denominado J48 [7][24]. As árvores de decisão [14] [42] [55] [56] são técnicas de aprendizagem de máquina supervisionada nas quais é criada uma árvore de decisão para a descoberta de padrões de classificação.

O C4.5 é um dos algoritmos mais conhecidos para construção de árvores de decisão a partir de um conjunto de dados de treinamento, além de ser uma evolução do algoritmo ID3, as principais diferenças entre eles são que apenas o algoritmo C4.5 consegue lidar com números contínuos e tratar valores desconhecidos.

Em uma árvore de decisão, os nós representam os nomes de atributos, os arcos os valores desses atributos e as folhas as classes a que pertencem cada exemplo. Conforme ilustrado na Figura 52, o conjunto de treinamento é constituído de um conjunto de exemplos formados por atributos, valores e uma classe associada. A árvore é criada a partir de um conjunto de exemplos de treinamento rotulados. Cada nó interno da árvore corresponde a um teste do valor de um atributo, os ramos dos nós são rotulados com os resultados possíveis do teste e as folhas possuem os valores rotulados do atributo classe (atributo a predizer) e cada caminho entre a raiz e uma folha é um padrão ou regra de classificação.

Neste exemplo, os valores dos atributos são dados de uma sessão de comunicação e a classe corresponde a um tipo de intrusão caracterizada por esse conjunto de valores. O algoritmo C4.5 constrói uma árvore de decisão a partir desses dados. A árvore pode ser representada por regras de condição-ação.

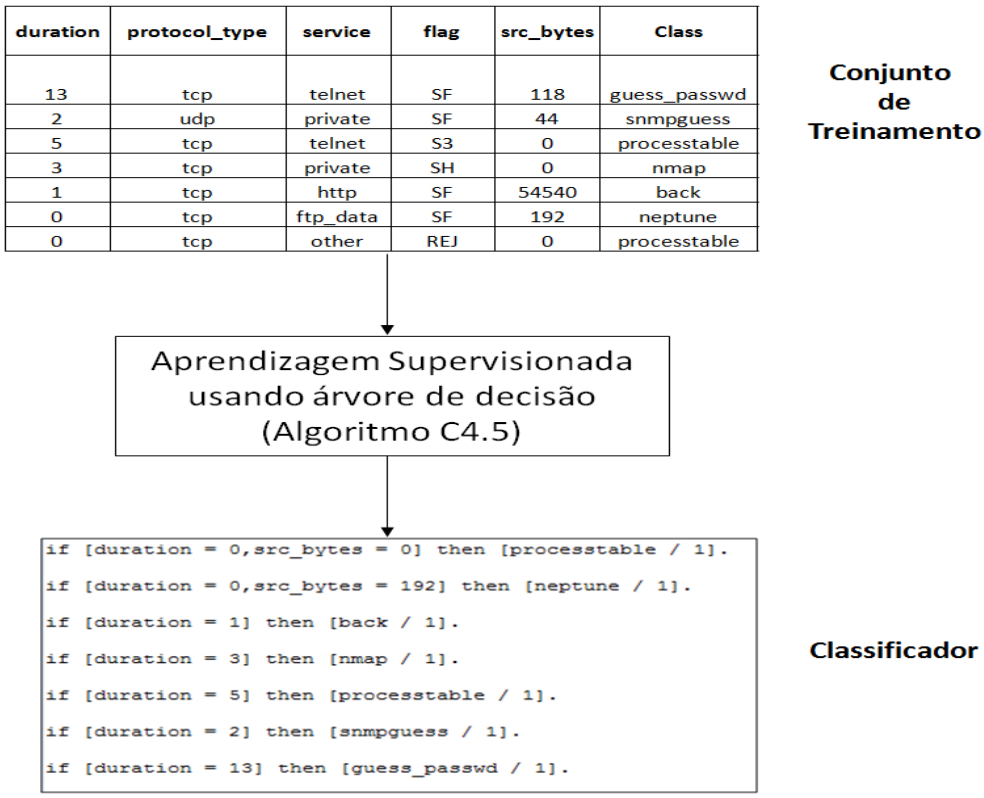


Figura 52. Funcionamento do algoritmo C4.5

O algoritmo C4.5 constrói a árvore de decisão utilizando uma abordagem "top-down" considerando qual atributo é o melhor para ser o nó raiz da árvore, isto é, qual atributo é mais significativo. Anteriormente quando se utilizava o algoritmo ID3, cada atributo era avaliado utilizando as medidas de Entropia e Ganho de informação. A entropia de um atributo é definida como [14]:

$$E(S) = -p1 \log_2 p1 - p0 \log_2 p0 \quad (4)$$

onde,

S é uma amostra de exemplos de treinamento;
p_1 *é* a proporção de exemplos positivos em S
p_0 *é* a proporção de exemplos negativos em S

S é uma amostra de exemplos de treino;
p_1 *é* a proporção de exemplos positivos em S
p_0 *é* a proporção de exemplos negativos em S

A entropia do conjunto de treinamento será zero quando todos os exemplos forem positivos e será 1 quando metade dos exemplos for positiva e metade negativa. Por exemplo, o conjunto de treinamento da Tabela 11 representa um conjunto de exemplos no qual alguém

decide se vai à praia (exemplos positivos) ou não (exemplos negativos).

Tabela 11. Conjunto de treinamento "Ir a praia"

Dia	Estado de saúde	Tempo	Classe
Final de semana	Boa saúde	Sol	sim
Final de semana	Doente	Sol	não
Feriado	Boa saúde	Sol	sim
Final de semana	Boa saúde	Sol	sim
Segunda-sexta	Boa saúde	Sol	não
Segunda-sexta	Doente	Chuva	não
Segunda-sexta	Doente	Sol	não
Segunda-sexta	Doente	Sol	não
Feriado	Boa saúde	Sol	sim
Final de semana	Boa saúde	Sol	sim

O valor "*final de semana"* do atributo "*Dia"* tem três possíveis valores, sendo um negativo e três positivos, a entropia seria a seguinte:

$$E(FinaldeSemana) = -(\frac{3}{4})\log_2\left(\frac{3}{4}\right) - (\frac{1}{4})\log_2(\frac{1}{4})$$

$$E(FinaldeSemana) = 0{,}811$$

A entropia do valor "feriado" do atributo "Dia" será 1, pois todos os exemplos são positivos e do valor "segunda-sexta" será 0, pois todos os exemplos são negativos.

O ganho de informação de um determinado atributo representa o quanto aquele atributo representa toda a função alvo, ou seja, a regra de classificação. Assim, os atributos com maior ganho de informação devem ser selecionados para sua inclusão em uma regra [8]. O ganho de informação é obtido subtraindo-se a entropia do atributo da soma ponderada da entropia de cada valor desse atributo, onde o peso é a quantidade de exemplos de cada valor que o atributo tem. Por exemplo, o peso do valor "feriado" é 0.2, pois existem dois valores "feriado" dentre os 10 exemplos. Assim, o ganho de informação para o atributo "*Dia"* é calculado da seguinte forma:

Ganho de informação = 1 – (0,4 * 0,811) – (0,4 * 1) – (0,2 * 0) = 0,2756 (5)

O ganho de informação seleciona como atributo-teste aquele que maximiza o ganho de informação. Assim, ele privilegia aqueles atributos com muitos valores possíveis. Por isso, no algoritmo C4.5 passou-se a utilizar a razão de ganho [74]:

$$razão\_de\_ganho(nó) = \frac{ganho}{entropia(nó)} \quad (6)$$

No cálculo da razão de ganho, primeiro é calculado o ganho de informação para todos os atributos e são considerados apenas aqueles que obtiveram ganho de informação acima da média, e em seguida são selecionados aqueles com maior razão de ganho [74]. Assim, a razão de ganho (algoritmo C4.5) se torna mais eficiente do que apenas calcular o ganho de informação (algoritmo ID3).

4.2.2 Resultados

A Tabela 12 mostra os resultados obtidos com o segundo estudo de caso. Na primeira coluna são listadas as experiências realizadas. A segunda coluna informa quantas percepções foram utilizadas como teste. A terceira coluna mostra a quantidade de exemplos de treinamento utilizado em cada uma das experiências. A quarta coluna informa a quantidade de intrusões classificadas corretamente ("true positive" + "true negative") e a quinta coluna mostra a quantidade de percepções classificadas incorretamente ("false positive" + "false negative"). Na última coluna é mostrada a acurácia ("true positive" + "true negative") / ("true positive" + "true negative" + "false positive" + "false negative").

Tabela 12. Valores de acurácia obtidos em cada experiência

Experiências	Conjunto de teste	Conjunto de Treinamento	Classificado Corretamente	Classificado Incorretamente	Acurácia
Exp. 1	50 exemplos	150 exemplos	37	13	74%
Exp. 2		500 exemplos	38	12	76%
Exp. 3		1000 exemplos	39	11	78%

4.2.3 Discussão

Observa-se nos resultados deste estudo de caso que quanto maior o tamanho do conjunto de treinamento maior a acurácia das regras aprendidas. Desta forma, para que o agente seja efetivo é necessário um conjunto de exemplos mínimo que deverá ser determinado de acordo aos requisitos de efetividade do agente que estiver sendo desenvolvido.

Os conjuntos de treinamento utilizados (150, 500 e 1000 instâncias, respectivamente) eram independentes entre si, pois o objetivo destas experiências era avaliar a variação na efetividade de acordo com o tamanho do conjunto de treinamento. No entanto, em uma aplicação real, o agente HyLAA deverá incrementar a quantidade de exemplos de treinamento e atualizar

constantemente sua base de conhecimento na medida em que aprende novas regras reativas. Desta forma, no quarto estudo de caso, o componente de aprendizagem do agente será avaliado sob esta perspectiva. Devido a aprendizagem das regras reativas pelo agente através da técnica de aprendizagem supervisionada, evitou-se o esforço de criação e a necessidade de um especialista no domínio para a atualização da base de conhecimento com as novas regras reativas aprendidas.

4.3 Estudo de caso 3: avaliação do agente híbrido com aprendizagem

O objetivo deste estudo de caso é avaliar a evolução da efetividade e desempenho do agente híbrido na medida em que ele aprende. Através do seu mecanismo de aprendizagem, o agente híbrido adapta-se ao seu ambiente e identifica novas intrusões não previamente definidas durante o seu projeto. Isto é feito através da indução de regras reativas considerando o realimentação das recomendações realizadas pelo agente através do seu componente RBC. O objetivo, método e resultados esperados neste estudo de caso são apresentados na Tabela 13. O agente processa 500 percepções representando 10 tipos diferentes de intrusões, usa a base de conhecimento ONTOID instanciada com 11 mil casos de intrusões correspondente ao RBC, sem nenhuma regra reativa pré-definida. A efetividade é avaliada em termo das medidas de "precision" e "recall", já introduzidas no estudo de caso 1.

Tabela 13. Organização do estudo de caso 3

	Objetivos	**Método**	**Resultados esperados**
Exp. 1	Avaliar a evolução na efetividade e desempenho do comportamento do agente híbrido na medida em que ele aprende novo comportamento reativo	Utilizar 500 percepções, uma base de conhecimento instanciada com 11 mil casos de intrusões e sem nenhuma regra reativa pré-definida	Melhor efetividade e desempenho do comportamento híbrido do agente na medida em que ele aprende novo comportamento reativo. Adaptabilidade do agente ao seu ambiente ao longo do tempo

4.3.1 Método

Neste estudo de caso, avalia-se a evolução na efetividade e desempenho do comportamento do agente híbrido na medida em que ele aprende novo comportamento reativo. Para isto, a cada 100 novas percepções o agente aprende um novo conjunto de regras reativas

utilizando essas percepções como conjunto de treinamento. As regras reativas aprendidas sempre substituem as anteriores, mas o conjunto de treinamento é mantido e incrementado a cada 100 novas intrusões percebidas. Assim, para o primeiro classificador serão usadas um conjunto de 100 exemplos, para o segundo, 200 exemplos e assim sucessivamente. Cada conjunto de 100 novas percepções mais o histórico de percepções do agente usado para aprendizagem é denominado de limiar de aprendizagem. No momento em que a quantidade de exemplos definido pelo limiar de aprendizagem é atingido (100 novos exemplos), novas regras reativas são aprendidas substituindo as anteriores e a efetividade do agente é avaliada em termo das medidas de "recall" e "precision".

Para avaliação do desempenho do agente foi utilizado o tempo de resposta. O tempo de resposta de um software corresponde à quantidade de tempo gasto desde uma requisição feita pelo usuário até que ela seja atendida pelo sistema [26][47]. Para o agente HyLAA, o tempo de resposta, corresponde ao tempo que ele levou desde que teve uma nova percepção até a recomendação de uma solução. Para medir o tempo de resposta do agente, foi utilizado o predicado "statistics" do LPA Prolog que fornece o tempo gasto pelo agente para processar uma determinada percepção. O computador utilizado para executar todas as experiências deste capítulo é o mesmo (Processador Intel Core I5 2.70 GHz, com 8 GB de memória RAM) para garantir que a configuração do computador não tenha impacto nos resultados da avaliação.

4.3.2 Resultados

A Tabela 14 apresenta os resultados da primeira experiência deste estudo de caso. A segunda linha da tabela apresenta o conjunto de percepções do agente, totalizando 500 novas percepções. A terceira linha mostra o tamanho do conjunto de treinamento utilizado para aprender cada novo classificador. O conjunto de treinamento é a soma das percepções anteriores do agente. A quarta linha informa a quantidade de regras aprendidas em cada limiar de aprendizagem. A quinta linha informa a média de "precision" (MP) e a média de "recall" (MR) obtida em cada limiar de aprendizagem. A última linha informa a média do tempo de resposta do agente para cada conjunto de 100 novas intrusões detectadas pelo agente.

Inicialmente, quando o agente é executado ele não possui regras na ONTOID que suportem o comportamento reativo, detectando as intrusões apenas através do raciocínio baseado em casos. O resultado da efetividade do agente ao detectar e responder as 100 primeiras intrusões é mostrado na segunda coluna e quinta linha da tabela (0,09 de média de "precision" e 0,0002 de média de "recall"). Na sexta linha é mostrado a média do tempo de resposta do agente

para detectar e responder as 100 primeiras intrusões (40.000 segundos).

No primeiro limiar de aprendizagem, o agente aprendeu 30 novas regras reativas, tornando-se assim híbrido. O resultado da efetividade do agente híbrido no primeiro limiar de aprendizagem é mostrado na terceira coluna e quinta linha da tabela (53% de média tanto de "precision" como de "recall"). Na sexta linha é mostrada a média do tempo de resposta agente nesse limiar de aprendizagem (1.680 segundos). No segundo limiar de aprendizagem, o agente aprendeu mais 74 regras reativas que substituíram as 30 regras reativas iniciais. O resultado da efetividade do agente híbrido neste limiar é mostrado na quarta coluna e quinta linha da tabela (68% de média de "precision" como de "recall"). Na sexta linha é mostrada a correspondente média de tempo de resposta do agente nesse limiar de aprendizagem (895 segundos). No terceiro limiar de aprendizagem, o agente aprendeu 81 novas regras reativas que substituíram as 74 regras reativas do segundo limiar de aprendizagem. O resultado da efetividade do agente híbrido neste limiar é mostrado na quinta coluna e quinta linha da tabela (76% de média de "precision" e "recall"). Na sexta linha é mostrado a média do tempo de resposta do agente para este limiar (500 segundos). No quarto limiar, usando como conjunto de treinamento as 400 percepções anteriores o agente também aprendeu 81 regras reativas que substituíram as 81 regras reativas anteriores. O resultado da efetividade da execução do agente com as 400 percepções anteriores mais as 100 novas percepções usando o conjunto de 81 regras reativas é mostrado na sexta coluna e quinta linha da tabela (81% de média de "precision" e "recall"). Na sexta linha é mostrado o tempo de resposta do agente para detectar as novas 100 intrusões usadas no quarto limiar de aprendizagem (500 segundos).

Tabela 14. Resultado da avaliação da efetividade do agente híbrido considerando diferentes limiares de aprendizagem

Limiar	**Sem aprendizagem**		**Primeiro limiar de aprendizagem**		**Segundo limiar de aprendizagem**		**Terceiro limiar de aprendizagem**		**Quarto limiar de aprendizagem**	
Percepções	100		100+100		200+100		300+100		400+100	
Conjunto de treinamento			100		200		300		400	
Classificador aprendido	-		30 regras		74 regras		81 regras		81 regras	
Efetividade (média de "precision" e"recall")	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**
	0,09	0,00	0,53	0,53	0,68	0,68	0,76	0,76	0,81	0,81
Média do tempo de resposta	40.000 segundos		1.680 segundos		895 segundos		500 segundos		500 segundos	

4.3.3 Discussão

Este estudo de caso avaliou a evolução da efetividade e tempo de resposta conforme o agente aprendia novo comportamento reativo. O agente obteve média de “precision” e “recall” muito baixas antes do agente tornar-se híbrido. Das 100 percepções apenas nove foram detectadas corretamente (Tabela 14). No entanto, a partir do primeiro limiar de aprendizagem (terceira coluna), onde um conjunto de 30 regras reativas foram aprendidas, tanto a média de “precision” quanto de “recall” aumentaram progressivamente. No último limiar, percebe-se que houve uma grande melhora na efetividade (média de “recall” e “precision”). O tempo de resposta do agente também foi melhorando progressivamente na medida em que ele aprendeu novo comportamento reativo. Assim, a partir dos resultados desta experiência, observa-se que o agente híbrido melhorou a efetividade e o tempo de resposta do seu comportamento na medida em que aprendeu novas regras reativas, tornando-se mais adaptado ao ambiente.

Os resultados deste estudo de caso mostraram que a efetividade e o tempo de resposta do agente melhoram gradativamente na medida em que ele aprende novo comportamento reativo, como pode ser observado na Tabela 14. No entanto, alguns fatores como a qualidade dos dados (ausência de inconsistências, dados duplicados, etc) e o algoritmo utilizado podem afetar estes resultados.

Neste estudo de caso, as médias de “precision” e “recall”, muitas vezes, têm o mesmo resultado. Nos sistemas de recuperação de informação tradicionais, normalmente, se recupera mais de um caso relevante para cada consulta e os resultados de “precision”, normalmente, são inversamente proporcionais aos de “recall”, ou seja, quando o valor de “precision” aumenta o de “recall” diminui e vice-versa. No entanto, o comportamento reativo afeta esta relação, pois, na maioria das vezes, existe apenas uma solução relevante para cada tipo de percepção. Assim, quando o agente age reativamente, no cálculo de “precision” e “recall”, o “número de casos relevantes na base de conhecimento” será igual a 1 (se a solução for relevante) ou 0 (se a solução não for relevante) e o “número de casos relevantes recuperados” também será igual a 1 ou a 0.

4.4 Estudo de caso 4: avaliação do agente híbrido RBC sem aprendizagem

Este estudo de caso avalia a efetividade e o desempenho do agente híbrido sem aprendizagem. Para isso foram desenvolvidas duas experiências, conforme mostrado na Tabela 15.

A primeira experiência avalia a efetividade em termo das medidas de "precision" e "recall" e o desempenho do agente híbrido sem aprendizagem através do tempo de resposta. O agente processa as mesmas 50 percepções do estudo de caso 1 usando a base de conhecimento ONTOID instanciada com 11 mil casos de intrusões e um conjunto de regras inseridas manualmente na base de conhecimento.

A segunda experiência avalia a efetividade do agente híbrido sem aprendizagem usando os mesmos dados de teste do estudo de caso 3. Nesta experiência, a efetividade é avaliada a cada 100 novas percepções processadas.

Tabela 15. Organização do estudo de caso 4

	Objetivos	**Método**	**Resultados Esperados**
Exp. 1	Avaliar a efetividadade e desempenho do agente híbrido sem aprendizagem com os mesmos dados de teste do estudo de caso 1	Executar o agente híbrido no ambiente com as mesmas 50 percepções do estudo de caso 1, usando três conjuntos diferentes de regras reativas capazes de detectar, respectivamente, 2, 6 e 10 tipos diferentes de intrusões Analisar os resultados da execução do agente híbrido sem aprendizagem	Melhor efetividade e desempenho quando se aumenta a quantidade de regras reativas para detecção de diferentes tipos de intrusões
Exp. 2	Avaliar a efetividadade do agente híbrido sem aprendizagem com os mesmos dados de teste do estudo de caso 3	Executar o agente híbrido no ambiente com as mesmas 500 percepções que foram usadas para o estudo de caso 3, usando três conjuntos diferentes de regras reativas capazes de detectar, respectivamente, 2, 6 e 10 tipos diferentes de intrusões Analisar os resultados da execução do agente híbrido sem aprendizagem	Melhor efetividade quando se aumenta a quantidade de regras reativas para detecção de diferentes tipos intrusões Menor evolução na efetividade ao longo do tempo do que o agente com aprendizagem (estudo de caso 3)

4.4.1 Método

Na primeira experiência, a avaliação da efetividade e desempenho do agente híbrido sem aprendizagem consiste na execução do agente com os mesmos dados de teste do estudo de caso 1

e base de conhecimento ONTOID instanciada com 11 mil casos de intrusões, utilizando o ponto de corte 0.9 e três conjuntos de regras reativas diferentes, inseridas manualmente na base de conhecimento. O primeiro conjunto de regras reativas é capaz de detectar dois tipos de intrusões dentre os 10 tipos de intrusões usados para o conjunto de percepções, o segundo conjunto de regras reativas é capaz de detectar 6 tipos de intrusões e o último conjunto de regras reativas é capaz de detectar todos os 10 tipos de intrusões. A efetividade é avaliada em termo das medidas de "precision" e "recall" e, para avaliar o desempenho, foi medido o tempo de resposta do agente, da mesma forma que no estudo de caso 3.

Na segunda experiência, a avaliação da efetividade do agente híbrido sem aprendizagem foi realizada através da execução do agente utilizando as mesmas 500 percepções utilizadas no estudo de caso 3 e da base de conhecimento instanciada com 11 mil casos de intrusões, utilizando o ponto de corte 0.9 e três conjuntos de regras reativas diferentes, inseridas manualmente na base de conhecimento. Assim como na primeira experiência, o primeiro conjunto de regras reativas é capaz de detectar dois tipos de intrusões dentre os 10 tipos de intrusões usados para o conjunto de teste, o segundo, 6 tipos diferentes de intrusões e o último, os 10 tipos de intrusões. Nesta experiência a efetividade do agente foi avaliada de forma incremental, ou seja, a cada 100 novas recomendações foi avaliada a efetividade de 100, 200, 300, 400 e 500 recomendações realizadas pelo agente. O objetivo foi analisar como a efetividade evolui ao longo do tempo.

4.4.2 Resultados

A Tabela 16 mostra os resultados obtidos com a primeira experiência deste quarto estudo de caso. A segunda coluna informa a quantidade de percepções, ressaltando que os exemplos foram distribuídos em igual quantidade entre os mesmos dez tipos diferentes de intrusões usadas no primeiro estudo de caso. A terceira coluna informa quais tipos de intrusões cada conjunto de regras reativas consegue detectar. Assim, as intrusões que as regras reativas não forem capazes de detectar serão tratadas pelo componente RBC do agente. As duas colunas seguintes informam, respectivamente, a média de "precision" e "recall" obtida pelo agente para detectar o conjunto de 50 percepções. A última coluna informa o tempo que o agente levou para detectar as 50 intrusões e recomendar uma solução ao administrador da rede para cada uma delas.

Tabela 16. Resultados da primeira experiência usando diferentes conjuntos de regras reativas e 50 percepções

	Percepções	Regras Reativas na ONTOID	Média Precision	Média Recall	Tempo de resposta
Exp. 1	As mesmas 50 percepções do estudo de caso 1	Conjunto de regras reativas capazes de detectar 2 tipos diferentes de intrusões	0,95	0,49	80.000 segundos
		Conjunto de regras reativas capazes de detectar a 6 tipos diferentes de intrusões	0,98	0,85	40.000 segundos
		Conjunto de regras reativas capazes de detectar 10 tipos diferentes de intrusões	1	1	5 segundos

As Tabelas 17 a 19 mostram a média de "precision" (MP) e a média de "recall" (MR) para 100, 200, 300, 400 e 500 percepções. A Tabela 17 informa a média de "precision" e "recall" obtidas quando o conjunto de regras reativas consegue detectar apenas dois tipos diferentes de intrusões ("apache" e "guess_password"), sendo os demais tipos de intrusões detectadas através do componente RBC. A Tabela 18 informa a média de "precision" e "recall" obtida quando o conjunto de regras reativas consegue detectar apenas 6 tipos diferentes de intrusões ("apache", "guess_password", "warezmaster", "satan", "smurf" e "back") e as demais através o componente RBC. A Tabela 19 informa a média de "recall" e "precision" obtidas quando o conjunto de regras reativas consegue detectar todos os 10 tipos diferentes de intrusões ("apache", "guess_password", "warezmaster", "satan", "smurf", "back", "portsweep", " ipsweep", "nmap", "pod").

Tabela 17. Resultado da avaliação do agente híbrido sem aprendizagem com um conjunto de regras reativas representando 2 tipos de intrusões

Conjunto de Percepções	**100**		**100+100**		**200+100**		**300+100**		**400+100**	
Exp. 2	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**
	0,32	0,32	0,32	0,32	0,45	0,35	0,42	0,35	0,51	0,35

Tabela 18. Resultado da avaliação do agente híbrido sem aprendizagem com um conjunto de regras reativas representando 6 tipos de intrusões

Conjunto de Percepções	**100**		**100+100**		**200+100**		**300+100**		**400+100**	
Exp. 2	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**
	0,84	0,84	0,90	0,90	0,92	0,92	0,93	0,93	0,94	0,94

Tabela 19. Resultado da avaliação do agente híbrido sem aprendizagem com um conjunto de regras reativas representando 10 tipos de intrusões

Conjunto de Percepções	**100**		**100+100**		**200+100**		**300+100**		**400+100**	
Exp. 2	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**
	0,97	0,97	0,97	0,97	0,98	0,98	0,98	0,98	0,98	0,98

4.4.3 Discussão

Os resultados da experiência 1 mostraram que o agente híbrido sem aprendizagem melhora tanto a efetividade quanto o tempo de resposta ao se aumentar o número de regras reativas que permitem detectar uma quantidade maior de intrusões diferentes (Tabela 16). Isto acontece porque, quando se aumenta o número de regras reativas que permitem detectar uma quantidade maior de intrusões diferentes, a acurácia deste conjunto de regras também aumenta, ou seja, elas detectam um número maior de intrusões diferentes corretamente e, consequentemente, o comportamento reativo ao usar este conjunto de regras com maior acurácia tem uma melhor efetividade. Por sua vez, o comportamento reativo é prioritário para o agente híbrido, ou seja, sempre que houver uma regra reativa que permita a detecção de uma intrusão, a mesma será utilizada sem que o agente precise usar o raciocínio baseado e casos. Assim, ao se aumentar o número de regras reativas que permitem detectar uma quantidade maior de intrusões diferentes, os tipos de intrusões que antes eram detectadas através do raciocínio baseado em casos, por similaridade, devido à ausência de regras reativas agora são detectadas de forma reativa. Já o tempo de resposta do agente melhora ao se aumentar o número de regras reativas porque o agente híbrido poderá detectar mais tipos de intrusões de forma reativa, usando regras de condição-ação que, normalmente, são processadas mais rápido do que usando RBC. Por exemplo, na primeira experiência deste estudo de caso, cada recomendação realizada de forma deliberativa levou em torno de 6,6 minutos, enquanto que a mesma recomendação realizada de forma reativa levou menos de 1 segundo para ser processada (Tabela 16).

Os resultados da experiência 2 também mostraram a relação direta entre a quantidade de regras reativas capazes de detectar diferentes tipos de intrusões e a efetividade do agente. Por exemplo, com o mesmo conjunto de 100 percepções iniciais (Tabela 17) obteve 32% de “precision” e “recall” ao usar um conjunto de regras reativas capaz de detectar 2 tipos de intrusões, 84% de “precision” e “recall” ao usar um conjunto de regras capaz de detectar 6 intrusões (Tabela 18) e 97% de “precision” e “recall” ao se usar um conjunto de regras capazes de detectar todos os 10 tipos intrusões (Tabela 19).

Em comparação aos resultados de efetividade da experiência do estudo de caso 3 (Tabela 14), o agente híbrido sem aprendizagem (Tabelas 17 a 19) obteve média de efetividade tanto inicial (100 primeiras intrusões detectadas) quanto final (500 primeiras intrusões detectadas) maior. No entanto, por não ter aprendizagem, houve uma menor variação ao longo do tempo nos valores de efetividade, uma vez que as regras reativas são pré-definidas na concepção do agente.

Por exemplo, o agente híbrido sem aprendizagem ao usar o conjunto de regras reativas capazes de detectar todos os 10 tipos diferentes de intrusões (Tabela 16), obteve uma variação na efetividade de apenas 1%. Já o agente com aprendizagem (Tabela 14), obteve uma efetividade inicial baixa ao detectar as 100 primeiras intrusões quando não tinha aprendido ainda nenhuma regra reativa, mas houve um aumento gradativo ao longo do tempo, aumentando 72% na média de "precision" e 80,98% de "recall" ao detectar as 500 intrusões quando já tinha aprendido um conjunto de 81 regras capazes de detectar os 10 diferentes tipos de intrusões.

Neste estudo de caso, todas as regras reativas tiveram que ser inseridas de forma manual na base de conhecimento durante a concepção do agente, representando um esforço relevante para o desenvolvedor, enquanto que o agente híbrido com aprendizagem aprendeu todas as regras de forma automática. Além disso, o conjunto de regras reativas do agente híbrido sem aprendizagem, por serem definidas na concepção do agente podem ficar desatualizadas com o tempo. Assim, quando houver mudanças no ambiente, será necessária a manutenção da base de conhecimento do agente, com atualização das regras reativas preexistentes e/ou a inserção de novas regras reativas para atender essas mudanças. Já no caso do agente híbrido com aprendizagem, a atualização das regras reativas, em muitos casos, será realizada sem custo de manutenção, através da aprendizagem. No entanto, para intrusões recentes, mesmo no caso do agente com aprendizagem, a base de conhecimento deverá ser atualizada manualmente ou através de algum mecanismo de atualização automática que obtenha dados de fontes externas.

4.5 Síntese

Os estudos de casos realizados neste capítulo serviram para avaliar a efetividade da arquitetura híbrida HyLAA, quanto a sua efetividade, desempenho, esforço de desenvolvimento e adaptabilidade através do desenvolvimento de quatro estudos de casos.

No primeiro estudo de caso realizado, avaliou-se a efetividade do componente RBC do agente experimentando-se diferentes pontos de corte (valores mínimos de similaridade) e atribuindo-se pesos maiores aos atributos mais relevantes. Através deste estudo de caso observou-se que o ponto de corte 0.9 é ideal para o agente HyLAA, isto é, ao recomendar casos com similaridade de aproximadamente 90% tem-se uma alta precisão das recomendações mantendo uma quantidade mínima de recomendações recuperadas ("recall"). Além disso, observou-se que a atribuição de pesos maiores aos atributos mais relevantes tem pouco impacto positivo na efetividade das recomendações. Assim, considerando o esforço necessário para

distribuir os pesos e atributos mais relevantes para todos os diferentes tipos de intrusões da base de conhecimento, é recomendável apenas em situações onde pequenos ganhos em efetividade forem muito relevantes. Em resumo, este estudo de caso alcançou os resultados esperados "Boa efetividade do componente RBC, ou seja, capacidade de recomendar soluções relevantes e ocultar informações irrelevantes" ao se usar o ponto de corte 0.9.

No segundo estudo de caso, avaliou-se a acurácia do componente de aprendizagem ao aprender novas regras reativas. Usando um conjunto de 1000 exemplos o agente conseguiu detectar corretamente 78% das intrusões. Neste estudo de caso, observou-se a relação direta entre o conjunto de treinamento e a acurácia das regras aprendidas. Assim, a acurácia das regras está diretamente relacionada ao conjunto de treinamento considerando-se a qualidade dos dados que compõem este conjunto de treinamento, isto é, a ausência de inconsistências, duplicidade, etc. Este estudo de caso alcançou o resultado esperado "Boa capacidade de detectar corretamente uma intrusão" ao se usar um conjunto de treinamento de 1000 exemplos.

No terceiro estudo de caso, avaliou-se a efetividade do agente híbrido com aprendizagem. Os resultados deste estudo de caso mostraram que a efetividade do comportamento do agente melhorou gradualmente na medida em que ele aprendia novas regras reativas. Em relação aos resultados do estudo de caso 4, o agente híbrido sem aprendizagem apresentou melhores resultados de efetividade do que o agente híbrido com aprendizagem, usando o mesmo conjunto de teste, no entanto, as regras reativas da base de conhecimento foram inseridas manualmente com um conjunto inicial de regras capazes de detectar, no mínimo, dois tipos diferentes de intrusões. Já o agente com aprendizagem foi desenvolvido sem nenhuma regra reativa predefinida e na medida em que ele interagia com o ambiente essas regras foram aprendidas automaticamente. Assim, o agente híbrido com aprendizagem ao perceber os resultados de suas ações no ambiente (positivo ou negativo), se auto-adapta as mudanças que ocorrem neste ambiente. Isso acontece através do aprendizado de comportamento reativo, mais rápido e eficiente. O uso deste tipo de arquitetura diminui a necessidade de manutenção evolutiva da base de conhecimento do agente, pois grande parte das mudanças que ocorrem no ambiente são aprendidas pelo agente. Considerando que o agente aprende continuamente, com o tempo a sua efetividade tende a se igualar a obtida pelo agente sem aprendizagem se a base de conhecimento do mesmo não for atualizada. Este estudo de caso alcançou os resultados esperados "Melhor efetividade e desempenho do comportamento híbrido do agente na medida em que ele aprende novo comportamento reativo" e "Adaptabilidade do agente ao seu ambiente ao longo do tempo".

No quarto estudo de caso, avaliou-se a efetividade e desempenho do agente híbrido sem

aprendizagem. Neste estudo de caso, observou-se que a efetividade e o desempenho do agente estão diretamente relacionados a quantidade de diferentes tipos de intrusões que as regras reativas são capazes de detectar. Por exemplo, ao usar um conjunto de regras que é capaz de detectar 2 dos 10 tipos diferentes de intrusões usadas no conjunto de teste, a efetividade média foi de 35%, com um conjunto de regras que é capaz de detectar 6 dos 10 tipos de intrusões usadas no conjunto de teste, a efetividade média foi de 94%, já usando um conjunto de regras reativas que é capaz de detectar 10 dos 10 tipos de intrusões, obteve-se uma efetividade média de 98%. O desempenho do agente também está diretamente relacionado a capacidade de detectar intrusões do conjunto de regras reativas. Por exemplo, ao usar um conjunto de regras capaz de detectar 2 de 10; 6 de 10 e 10 de 10 tipos de intrusões com um conjunto de teste de 50 exemplos, obteve-se, respectivamente, o tempo de resposta média de 80000, 40000 e 5 segundos. Este estudo de caso alcançou os resultados esperados "Melhor efetividade quando se aumenta a quantidade de regras reativas para detecção de diferentes tipos intrusões" e "Menor evolução na efetividade ao longo do tempo do que o agente com aprendizagem";

A partir dos resultados dos quatro estudos de casos realizados, conclui-se que o agente híbrido com aprendizagem utilizando a arquitetura HyLAA é mais efetivo do que um agente com comportamento deliberativo quando executado independentemente por apresentar um melhor balanço entre efetividade e desempenho. Já em relação ao agente híbrido sem aprendizagem, as vantagens são a aprendizagem automática de regras e a adaptabilidade as mudanças do ambiente.

5. GENERALIZANDO HYLAA EM UMA ARQUITETURA DE REFERÊNCIA

Este capítulo propõe uma arquitetura de referência chamada HyLARA ("Hybrid and Learning Agent Reference Architecture") baseada na generalização dos conceitos da arquitetura específica HyLAA, definida no capítulo anterior. A arquitetura HyLARA que especifica uma solução de arquitetural genérica para o desenvolvimento de agentes de software híbridos com aprendizagem. Assim, a partir da arquitetura HyLARA, diversas arquiteturas concretas podem ser construídas, especializando os seus componentes e relacionamentos genéricos.

Uma arquitetura de referência de software [13] especifica uma solução arquitetural genérica para o desenvolvimento de arquiteturas de software específicas. Ele inclui componentes comuns a todas as arquiteturas de software e seus relacionamentos, um vocabulário comum, uma metodologia de mapeamento da arquitetura de referência para uma arquitetura específica e boas práticas de projeto.

A arquitetura HyLARA é baseada em ontologias, isto é, a base de conhecimento do agente deve ser representada em ontologias. As ontologias são ideais para a construção de bases conhecimento por vantagens, como maior expressividade semântica, descrição formal do conhecimento, reusabilidade, facilidade de adaptação e integração.

Este capítulo está organização em quatro seções. A seção 5.1 apresenta uma visão geral da arquitetura HyLARA. A seção 5.2 mostra cada um dos componentes da arquitetura HyLARA exemplificados no domínio da segurança da informação. A seção 5.3 traz um guia de mapeamento da arquitetura de referência HyLARA para uma arquitetura concreta.

5.1 Visão geral da arquitetura HyLARA

A HyLARA é uma arquitetura de referência, ou seja, nela foram definidas apenas os componentes e conceitos genéricos, comuns a todos os agentes a serem desenvolvidos. Assim, agentes de software híbridos (com diferentes formas de raciocínio, representação do conhecimento e técnicas de aprendizagem), podem ser desenvolvidos a partir da mesma. Para demonstrar a aplicabilidade da arquitetura HyLARA, foi desenvolvido um agente híbrido (seção 3.4) baseado nessa arquitetura. A Figura 53 ilustra a relação de especialização entre uma realização (arquitetura concreta) e a arquitetura HyLARA.

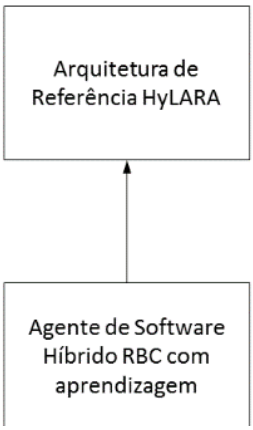


Figura 53. Realização da arquitetura de Referência HyLARA através do desenvolvimento de um agente híbrido RBC com aprendizagem

A Figura 54 ilustra os componentes e relacionamentos da arquitetura HyLARA, em uma visão estática. Ela é composta de dois componentes principais: o componente de desempenho ("Performance") e o componente de aprendizagem ("Learning").

O componente de desempenho é estruturado conforme a arquitetura de um agente genérico (Figura 2): percebe, mapeia percepções para ações e age sobre o ambiente. O mapeamento das percepções para ações pode ser realizado tanto por simples regras reativas (Figura 3) quanto por um processo de raciocínio (Figura 4). O componente de aprendizagem estrutura o mecanismo de aprendizado do agente baseado em técnicas de aprendizagem de máquina.

O componente "Performance", possui os seguintes subcomponentes: "Perception interpretation", "Reactive System", "Ontology-based Knowledge Base" e "Deliberative System". O subcomponente "Perception interpretation", é responsável por identificar o tipo de percepção, isto é, se é uma nova percepção ou uma percepção de realimentação. Uma percepção de realimentação, é o resultado de uma ação no ambiente. Por exemplo, após realizar uma ação "Bloquear a porta de comunicação 21", a percepção de realimentação poderia ser "Configuração realizada com sucesso" ou "Falha na configuração". O subcomponente "Reactive system", é o responsável por realizar o comportamento reativo do agente que, normalmente, é composto por um conjunto de regras de condição-ação. O subcomponente "Deliberative System" é o responsável pelas ações que necessitam de algum processo de raciocínio para serem realizadas, normalmente essas ações só são executadas quando não há uma ação reativa, pois tem um custo computacional maior. O subcomponente "Ontology-based knowledge base" é a base de

conhecimento do agente, representada por uma ontologia de aplicação.

O componente “Learning” é responsável por implementar melhorias no comportamento do agente RBC. Ele é constituído pelos dois subcomponentes: “Critic” e “Learning”. O “Critic” avalia o “realimentação” como o resultado das ações do agente de acordo com um padrão de desempenho. O componente “Learning” é responsável por fazer as melhorias de comportamento do agente utilizando uma determinada técnica de aprendizagem. Ele usa o realimentação do “Critic” para melhorar as ações realizadas pelo componente de “Performance”.

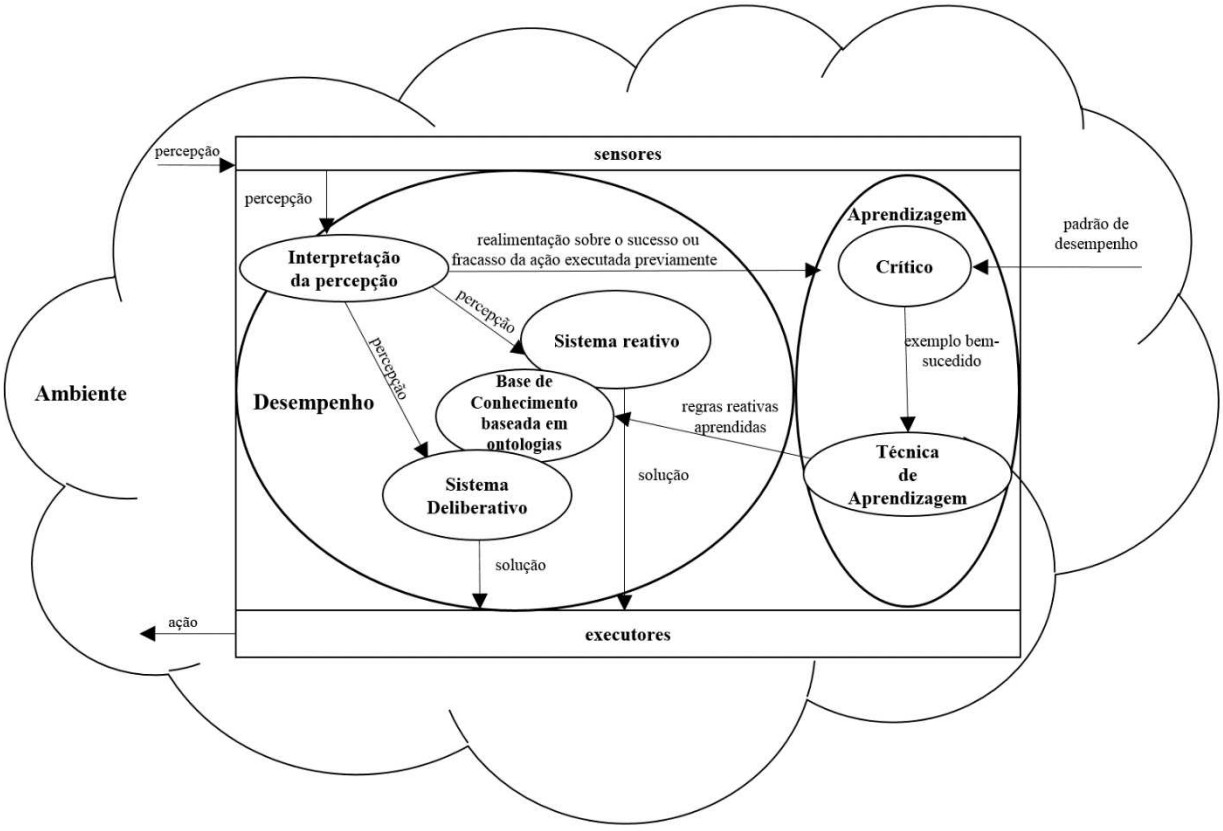


Figura 54. A arquitetura de referência HyLARA

A Figura 55 ilustra o diagrama de estados genéricos por que passa um agente desenvolvido segundo a arquitetura HyLARA, ou seja, uma visão dinâmica. O estado inicial do agente é “Monitoring”. Neste estado o agente monitora o ambiente à espera de uma percepção que pode ser tanto uma nova percepção quanto o resultado de uma ação já executada no ambiente (“realimentação”).

Quando o agente tem uma nova percepção, passa para o estado “Processing for a reactive solution”, e se encontrar uma ação reativa para essa percepção, a ação é executada no ambiente e esse agente volta ao estado “Monitoring”, mas, caso não encontre, passa para no estado “Processing for a deliberative solution”. Nesse estado, se encontrar uma ação deliberativa, essa ação é executada no ambiente e o agente volta ao estado “Monitoring”.

Quando o agente tem uma percepção de realimentação, ele passa para o estado “Processing successful behaviours”, no qual o agente avalia se a ação teve bom resultado ao ser executada e esse resultado é então armazenado na base de conhecimento para ser utilizado posteriormente para aprendizagem. Para sair do estado “Processing successful behaviours” a quantidade de ações bem avaliadas deve atingir um limiar predeterminado. Ao atingir esse limiar, o agente vai para o estado “Learning”, onde é realizado o aprendizado de novas regras reativas. No entanto, o agente continua, paralelamente, percebendo novas intrusões e armazenando as ações que foram bem sucedidas. Após gerar as novas regras e atualizar o subcomponente “Reactive System”, o agente volta ao estado “Monitoring”. O agente também pode ter múltiplos estados paralelos dependendo do domínio de aplicação, por exemplo, quando o agente tem que tratar várias percepções ao mesmo tempo.

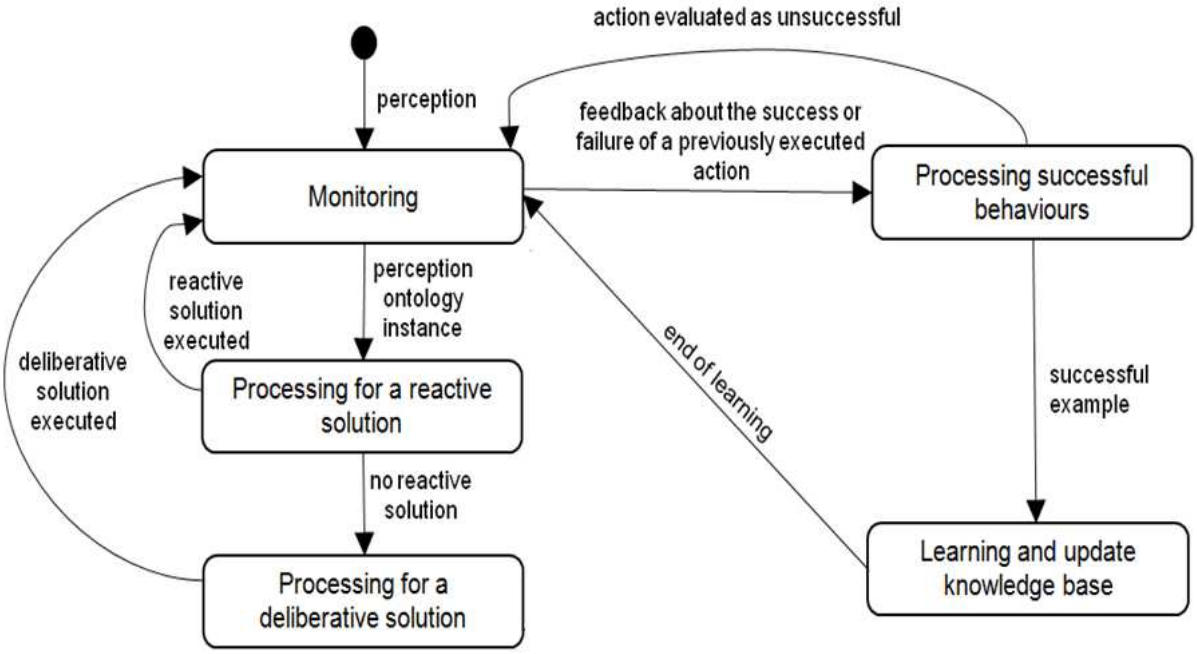


Figura 55. Diagrama de estados da arquitetura HyLARA

5.2 Componentes da arquitetura HyLARA

Nas próximas seções, os componentes de desempenho (“Performance”) e aprendizagem (“Learning”) da arquitetura HyLARA são apresentados. O componente de desempenho representa um agente de software híbrido e o componente “Learning” implementa uma determinada técnica de aprendizagem de máquina, tornando o agente capaz de melhorar o seu comportamento ao longo do tempo.

Algumas características de mapeamento destes componentes estão relacionadas ao ambiente que o agente estará inserido (classificação disponível na seção 2.3), por isso, é fundamental antes de realizar o mapeamento analisar as características deste ambiente. Por exemplo, a arquitetura HyLARA, por contemplar a aprendizagem de novo comportamento reativo é adequada para ambientes dinâmicos, já para ambientes estáticos o ideal seria uma arquitetura reativa ou deliberativa.

5.2.1 Componente de desempenho

O componente de desempenho (“Performance”) ilustrado na Figura 54 representa um agente híbrido composto pelos subcomponentes reativo (“Reactive System”), deliberativo (“Deliberative System”) e pela base de conhecimento (“Ontology-based knowledge base”). Normalmente, os agentes realizam o mapeamento de uma percepção ou de um conjunto de percepções para uma ação de forma reativa ou deliberativa. Esse mapeamento é denominado reativo quando uma percepção é mapeada diretamente para uma ação. No entanto, quando para encontrar uma ação para uma determinada percepção é necessário um processo de raciocínio o mapeamento é considerado deliberativo. Um terceiro tipo de mapeamento é o híbrido, o qual é o adotado neste trabalho. Nesse tipo de mapeamento, dada uma determinada percepção ou conjunto de percepções o agente pode realizar o mapeamento tanto de forma reativa quando deliberativa.

O comportamento reativo está relacionado ao conhecimento bem estabelecido e previsível ou ao conhecimento aprendido. Por exemplo, um agente cujo domínio de aplicação for um jogo, normalmente, terá um conjunto de regras bem estabelecidas, que podem ser reativas. Em um jogo de xadrez, uma regra reativa poderia ser “se controla as peças brancas então inicia o jogo”. Esse conjunto de regras conhecidas e bem estabelecidas devem ser incluídas no componente reativo do agente. Já o conhecimento utilizado pelo comportamento deliberativo é segmentado e, muitas vezes, incompleto.

O comportamento deliberativo está relacionado ao processo de busca por uma solução, isso acontece através de algum mecanismo de raciocínio e pode envolver algum tipo de planejamento. A busca da solução é realizada na base de conhecimento do agente por um mecanismo de inferência. O comportamento híbrido torna o agente mais flexível, de forma que ele consegue agir tanto reativamente quanto deliberativamente.

As Figuras 56 e 57 ilustram, de forma simplificada, o funcionamento do componente de "Performance" da arquitetura HyLARA implementado em Prolog. Supondo que um agente tem uma nova percepção cujo objetivo é descobrir se Julie e Bob são primos e que não há nenhuma regra no componente "Reactive system" (Figura 56) que responda diretamente que Julie e Bob são primos, será realizada uma inferência através do componente "Deliberative System" e através da regra 4 (em negrito), poderá se descobrir que eles são primos uma vez que seus pais são irmãos (Michael e Lily). No entanto, um processo de inferência normalmente é computacionalmente mais caro do que do que um comportamento reativo. Assim, se for necessário que o agente responda muitas vezes se Julie e Bob são primos é conveniente criar uma regra reativa, assim como foi criada no exemplo da Figura 57 ("**cousins(bob,julie).**").

Deliberative system

```
parent(X,Y) :- father(X,Y).
parent(X, Y) :- mother(X,Y).
brothers(X,Y):- parent(X,Z),parent(Y,Z).
cousins(X,Y):-brothers(A,B),parent(X,A),parent(Y,B).
```

Reactive system

```
father(michael,james).
father(lily,james).
father(bob,michael).
mother(michael,mary).
mother(lily,mary).
mother(julie,lily).
```

Figura 56. Exemplo do componente "Performance"

Deliberative system

parent(X,Y) :- father(X,Y).
parent(X, Y) :- mother(X,Y).
brothers(X,Y):- parent(X,Z),parent(Y,Z).
cousins(X,Y):-brothers(A,B),parent(X,A),parent(Y,B).

Reactive system

father(michael,james).
father(lily,james).
father(bob, michael).
mother(michael,mary).
mother(lily,mary).
mother(julie,Lily).
cousins(bob,julie).

Figura 57. Exemplo do componente "Performance" com uma nova regra reativa aprendida

As regras reativas podem ser criadas manualmente, sendo necessário um especialista no domínio para criá-las. Quando o especialista tem um conhecimento substancial do domínio e o ambiente não muda ao longo do tempo as regras reativas tendem a ser bastante efetivas. No entanto, esse processo pode ser bastante custoso no que se refere ao esforço de criar as regras se o ambiente do agente for muito dinâmico. Uma boa solução para esse problema é a automatização do processo de criação de regras reativas através da aprendizagem. Essa abordagem elimina o esforço de criação manual, o que, dependendo do domínio pode ser muito vantajoso. No entanto, essas regras podem não ser tão efetivas quanto as criadas por um especialista no domínio. Para resolver esse problema, normalmente, se realizam diversos testes para verificar se as regras aprendidas são efetivas.

5.2.2.1 Sistema reativo

O objetivo do componente reativo é fornecer o comportamento inteligente através de um conjunto de comportamentos simples e rápidos. Neste componente, o agente não realiza qualquer tipo de raciocínio. O sistema reativo contém um conjunto de regras do tipo <perception, action> representando pares de problemas bem conhecidos e soluções correspondentes sobre um domínio. Para cada percepção satisfazendo a condição, uma ação correspondente na base de conhecimento é selecionada e executada. Por exemplo, se a base de conhecimento tem a regra reativa "if (pessoa1 = "Bob") and (pessoa2 = "Julie") then "Answer: They are cousins" e a percepção tem os mesmos valores da condição da regra ("Bob" e "Julie") então a ação da regra ("Answer: They are cousins") poderá ser utilizada como ação do agente.

Na arquitetura HyLARA, o sistema reativo pode ter tanto a mesma estrutura do agente reativo simples (Figuras 2.2 e 2.3 da fundamentação teórica) quanto a estrutura do agente reativo com estado (Figuras 2.6 e 2.7 da fundamentação teórica). A escolha de um ou outro tipo de agente para o sistema dependerá dos requisitos do domínio no qual o agente estará inserido. Assim, se não houver necessidade de manter o estado do ambiente, o tipo reativo simples poderá ser utilizado. No entanto, se for necessário manter um histórico das percepções e ações anteriores do agente, o tipo reativo com estado deverá ser utilizado, pois esse tipo de agente atualiza constantemente o seu estado a partir das novas percepções do ambiente e suas ações são executadas de acordo com essa atualização. Por exemplo, um agente cujo objetivo seja jogar xadrez contra um adversário humano deverá obrigatoriamente guardar o seu histórico de ações, pois as suas ações futuras serão influenciadas por elas. No entanto, um agente cujo objetivo é manter uma única sala limpa, não precisa ter conhecimento sobre o estado anterior da sala, mas apenas o estado atual, limpa ou suja. Dessa forma, durante o mapeamento do sistema reativo genérico da arquitetura HyLARA para um sistema reativo a ser utilizado por um agente real, todos os requisitos e características do agente devem ser analisados.

5.2.2.2 Sistema deliberativo

Um agente deliberativo é qualquer agente que tenha capacidade de encontrar uma solução para um problema por meio de um processo de raciocínio, seja por raciocínio dedutivo, raciocínio prático (agentes BDI), raciocínio por analogia, raciocínio baseado em casos, entre outros.

Um agente baseado em metas mantém o estado do ambiente e tem uma meta a ser alcançada. Para isso, deve executar uma ação ou uma sequência de várias ações determinadas através de um mecanismo de raciocínio sobre a base de conhecimento do agente. Por isso, normalmente, eles são menos eficientes do que os agentes reativos, levando mais tempo para encontrar a ação apropriada a uma nova percepção. A estrutura de um agente deliberativo da arquitetura HyLARA é a mesma do agente baseado em objetivos (Figuras 9 e 10 do capítulo de fundamentação teórica), ou seja, ele é composto por um mecanismo de raciocínio e por uma base de conhecimento.

Um mecanismo típico de raciocínio é o dedutivo. Ao usar o raciocínio dedutivo, a representação simbólica do domínio do agente em uma base de conhecimento será necessária. Essa base de conhecimento representará os conceitos, fatos e regras do domínio em que o agente estará inserido. O agente deliberativo para encontrar uma ação adequada a uma determinada percepção irá realizar um processo de inferência usando o motor de inferência próprio da

linguagem lógica em que foi implementado. Esse processo de inferência consiste em uma estratégia de busca usando os fatos e regras da base de conhecimento.

O componente deliberativo da arquitetura HyLARA, chamado de "Deliberative System" poderá ser mapeado para qualquer tipo de raciocínio, tais como raciocínio dedutivo, indutivo e por analogia dependendo dos requisitos do agente e das características do seu domínio.

Uma questão essencial para definir qual tipo de raciocínio o agente deve utilizar é analisar se há conhecimento disponível acerca do domínio e se esse conhecimento se esse conhecimento é bem estruturado e confiável. Por exemplo, quando o domínio não é completamente conhecido, é recomendável o uso do raciocínio baseado em casos, uma vez que a base de conhecimento é construída de forma incremental através da instanciação de novos casos compostos representando um determinado problema e a sua respectiva solução. Já quando há uma quantidade de conhecimento suficiente acerca do domínio, de onde se obtenha novas informações por meio de inferências pode-se utilizar, por exemplo, o raciocínio indutivo.

5.2.2.3 Base de conhecimento

Os agentes de software têm dois tipos básicos de conhecimento: o seu conhecimento interno e o conhecimento compartilhado com outros agentes. O conhecimento interno é aquele necessário apenas para realização de suas próprias ações e o conhecimento compartilhado é aquele comum a todos os agentes da sociedade e/ou entidades externas.

A base de conhecimento de um agente pode incluir o conhecimento do mundo (domínio), o seu histórico de percepções (memória) e as regras de mapeamento de percepção para ação (regras reativas ou de inferência). Formas populares de representação da base de conhecimento do agente é a representação simbólica expressa em lógica e através de ontologias. Para isso, são utilizadas frequentemente linguagens simbólicas, como o Prolog e Jess, onde o conhecimento é representado através de fatos e regras.

Uma ontologia, para ser processável por um sistema de computação deve ser especificada em uma determinada linguagem de representação. Duas formas comuns de representação de ontologias são os frames e a OWL ("Web Ontology Language") [43], sendo que a segunda é a mais atual e mais utilizada atualmente.

Formalmente, uma ontologia pode ser definida como uma tupla O = (C,H,I,R,P,A) [20] onde,

$C = C_C \cup C_I$ é o conjunto de entidades da ontologia. São designados por um ou mais termos em linguagem natural. O conjunto C_C é composto pelos identificadores das classes, isto é, conceitos que representam entidades que descrevem um conjunto de objetos (por exemplo, "professora" $\in C_C$) enquanto o conjunto C_I é constituído por identificadores de instância (por exemplo, "instância Anne Smith" C_I), descrevendo objetos, que são entidades únicas.

$H: C_C \times C_C;\ H = \{(c_1, c_2) | c_1 \in C_C \wedge\ c_2 \in C_C\}$ é o conjunto de relacionamentos taxonômicos entre conceitos, que definem uma hierarquia de conceitos, as quais significam que c_1 é uma subclasse de c_2, por exemplo, "(professora, pessoa)".

$I: C_C \times C_I;\ I = \{(c_1, c_2) | c_1 \in C_C \wedge\ c_2 \in C_I\}$ é o conjunto das relações entre os identificadores de classes e os identificadores de instâncias correspondentes, exemplo "(Person, "instância Anne Smith ")".

$rel_k: R \rightarrow P(C);\ R = \{rel_k\ (c_1, c_2, \ldots, c_n)\ |\ i, c_i\ \in\ C_C\} \cup \{rel_k\ (c_1, c_2, \ldots, c_n)\ | i, c_i\ \in\ C_I\}$ é o conjunto de relacionamentos não taxonômicos de uma classe da ontologia, por exemplo, "mother_of("instância Anne Smith", "instância Kate Smith")".

$prop_k : P \rightarrow C_C \times DataType;\ P = \{prop_k(c_i, datatype) | c_i\ \in\ C_I\} \cup \{prop_k(c_i, value) |\ c_i\ \in\ C_I$é o conjunto de propriedades de uma classe da ontologia e seus tipos de dados básicos, por exemplo, "data_de_aniversario (Pessoa,data)" e instâncias correspondentes, por exemplo "data_de_aniversario("instancia Anne Smith", 06/07/2000)".

$rule_k : R \rightarrow P(C)$ é um conjunto de axiomas, regras que permitem verificar a consistência de uma ontologia e obter novos conhecimentos através da inferência; $rule_k$ é uma regra do tipo $condição_x\ \ conclusão_y\ onde, condição_x = (cond_1, cond_2, \ldots, cond_n) |\ z, cond_z \in H \cup I \cup R \cup P\}$. Por exemplo, " $Pessoa_1,\ Pessoa_2, Pessoa_3, mãe\_de(Pessoa_1, Pessoa_2) \wedge mãe\_de\ (Pessoa_1, Pessoa_3)\ \ irmão\_de\ (Pessoa_2, Pessoa_3)$" é um axioma que indica que se duas pessoas têm a mesma mãe então, elas são irmãs.

Ontologias [25][26] são estruturas de representação do conhecimento capazes de expressar um conjunto de entidades em um determinado domínio, os seus relacionamentos e axiomas, sendo usadas por sistemas baseados em conhecimento como base de conhecimento para representar e compartilhar o conhecimento de um domínio de aplicação particular. Esses sistemas permitem que o processamento semântico de informações e uma interpretação mais precisa dos dados, proporcionando maior efetividade na recuperação de informações do que os

sistemas tradicionais de informação. Além, disso as ontologias são genéricas, permitindo com que sejam reusadas e implementadas em diversas linguagens de programação. Nesse contexto, arquitetura HyLARA recomenda o uso de ontologias para representação da base de conhecimento dos agentes de software desenvolvidos.

As ontologias se classificam em ontologias de domínio, tarefas e aplicação (Figura 21 descrita no capítulo de fundamentação teórica). Assim, o mapeamento da base de conhecimento genérica da arquitetura HyLARA para uma base de conhecimento concreta a ser utilizada pelo agente, deverá ser uma ontologia de aplicação. Essa ontologia poderá ser construída do zero ou reusada.

5.2.2.4 Sistema de aprendizagem

Uma das características mais importantes de um agente de software é a aprendizagem, pois o torna mais efetivo nas suas ações freqüentes. Muitas técnicas de aprendizagem, como aprendizagem supervisionada, aprendizagem não-supervisionada e aprendizagem por reforço têm sido utilizadas para que o agente tenha a capacidade de aprender.

Mitchell [46] define que um programa aprende, a partir da experiência E, em relação a uma classe de tarefas T, com medida de desempenho P, se seu desempenho em T, medido por P, melhora com E. Em outras palavras, um programa aprende se seu desempenho melhora a partir da experiência na realização de uma determinada tarefa.

O componente de aprendizagem da arquitetura HyLARA é responsável pela execução de melhorias no comportamento do agente a partir da observação do resultado das suas ações no ambiente. Na medida em que o agente observa que suas ações deliberativas freqüentes foram bem-sucedidas ele aprende novo comportamento reativo. O resultado esperado dessa abordagem é um agente mais efetivo, ou seja, que melhora o seu processo de decisão, realizando ações que tenham mais sucesso e com melhor desempenho.

A aprendizagem de novo comportamento também possibilita que o agente se adapte automaticamente ao seu ambiente. Essa característica é essencial para ambientes dinâmicos, onde mudanças ocorrem frequentemente, pois sem a aprendizagem de novos comportamentos o agente deve ser reprogramado a cada nova mudança do ambiente, gerando altos custos de manutenção.

5.3 Mapeando a arquitetura de referência para uma realização

HyLARA é uma arquitetura de referência e, por tanto, genérica. Assim, para o seu uso por um agente todos os seus componentes devem ser mapeados para componentes concretos alinhados aos requisitos particulares do agente que estiver sendo desenvolvido. Nas próximas subseções as diretrizes gerais de mapeamento de cada componente da arquitetura HyLARA são apresentadas.

5.3.1 Mapeamento do componente de desempenho

O componente de Desempenho (“Performance”) da arquitetura HyLARA representa um agente híbrido sem aprendizagem composto pelo sistema deliberativo, reativo e pela base de conhecimento. O único requisito geral para uso do componente de Desempenho para desenvolvimento de uma arquitetura de agente concreta é que o agente deva ser híbrido, uma vez que a HyLARA contempla apenas o desenvolvimento desse tipo de agente.

Nas próximas seções as diretrizes para mapeamento do sistema deliberativo, reativo e da base de conhecimento genéricos para componentes concretos são descritas.

5.3.1.1 Sistema deliberativo

O primeiro passo para definir o sistema deliberativo da arquitetura concreta é definir o mecanismo de raciocínio que será utilizado. Para isso, deve-se verificar as características do domínio, os requisitos funcionais do agente e o seu ambiente. Por exemplo, quando as regras de domínio genéricas não são completamente conhecidas, o uso de raciocínio baseado em casos é recomendado, porque, o conhecimento de domínio é adquirido de forma incremental. Caso contrário, quando o domínio de conhecimento é estável, o raciocínio dedutivo pode ser usado. Em seguida, os requisitos não-funcionais, tais como o desempenho da aplicação deve ser analisado. Por exemplo, se a aplicação exige alto desempenho, o raciocínio baseado em casos pode ser impraticável devido ao tempo gasto no processamento.

5.3.1.2 Sistema reativo

Ao realizar o mapeamento do sistema reativo genérico da arquitetura HyLARA para uma aplicação específica deve ser decidir entre adquirir todas as regras reativas através da aprendizagem ou, inicialmente usar um conjunto de regras reativas estáticos definidos manualmente por um Engenheiro do Conhecimento e ao longo do tempo realizar a aprendizagem

de novas regras. Esta escolha pode ser com base nas características de domínio de aplicativo e nos seus requisitos não-funcionais. Por exemplo, se houver um conhecimento bem estabelecido sobre o domínio é recomendado o uso de regras estáticas iniciais.

5.3.1.3 Base de conhecimento

Na arquitetura HyLARA, o uso de ontologias é obrigatório para representação base de conhecimento para todas as arquiteturas específicas derivadas devido a inúmeras vantagens de ontologias. Por exemplo, eles têm expressividade semântica e seus conceitos são semanticamente relacionados permitindo pesquisas e inferências eficazes, facilitando assim a sua reutilização. Assim, o componente "Ontology-based knowledge base" deve ser mapeado para uma ontologia aplicação. A ontologia de aplicação representa tanto os conceitos de um determinado domínio quanto às tarefas de associadas a esse domínio.

A ontologia de aplicação poderá ser reusada parcial ou integralmente se houver ontologias do mesmo domínio disponíveis para reuso ou completamente desenvolvida, caso contrário. Para guiar o desenvolvimento da ontologia é recomendável utilizar-se de metodologias e ferramentas que facilitem o esforço de especificação e implementação da ontologia. No contexto do grupo GESEC [17], há a técnica GAODT [61] e a ferramenta GAODT Tool [61] que apóiam a criação da ontologia de aplicação, o processo Ontojoin [64] e a ferramenta Ontojoin Tool [64] que apoiam a integração de ontologias, o processo APPONTO [62] e a ferramenta DIPPAO [62] que suportam o povoamento automático de ontologias.

5.3.1.4 Mapeamento do sistema de aprendizagem

O sistema de aprendizagem deve ser mapeado para uma abordagem particular de aprendizagem supervisionada ou aprendizagem por reforço. Para escolher a abordagem de aprendizagem mais adequada para um determinado agente deve-se observar seu ambiente. Por exemplo, para usar a aprendizagem supervisionada é necessário um conjunto de treinamento inicial definido disponível. Por outro lado, para a aprendizagem de reforço, um realimentação, positivo ou negativo, das ações do agente deve ser fornecido. Em seguida, a técnica de aprendizagem mais adequada e algoritmo que irá implementá-lo deve ser escolhido. Por exemplo, quando se utiliza aprendizado supervisionado, as árvores de decisão são uma técnica de aprendizagem comum e uma das maneiras mais eficientes para implementar uma árvore de decisão é usar o algoritmo C4.5. O componente crítico é o responsável pela avaliação do realimentação do ambiente, isto é, ele determina se uma ação foi ou não bem-sucedida no

ambiente. Para realizar essa avaliação o crítico usa um padrão de desempenho. O padrão de desempenho é formado por um conjunto de regras que representam quais os resultados aceitáveis das ações do agente. Por exemplo, para um agente que realize as ações de monitorar um perfil de consumidor web e recomendar produtos de acordo a esse perfil poderia ter um atrasado de alguns minutos. O componente de aprendizagem é responsável pelas melhorias no comportamento do agente ao longo do tempo. Esse componente também deverá ser especializado considerando os requisitos funcionais e não-funcionais do agente que está sendo desenvolvido.

5.4 Síntese

Este capítulo apresentou a arquitetura híbrida de referência HyLARA, primeiro a arquitetura foi apresentada em uma perspectiva estática contendo uma visão geral da arquitetura e a descrição de cada um dos componentes individualmente e, logo após, uma visão dinâmica do funcionamento da arquitetura, mostrando o seu funcionamento completo, ilustrado através dos diagramas de atividade e estado. Também foram apresentadas as principais diretivas de mapeamento dos componentes genéricos da arquitetura para componentes particulares a um determinado domínio de aplicação.

6. CONCLUSÕES

O aumento da complexidade das aplicações de software tem requerido a adoção de novos paradigmas de desenvolvimento. Os agentes de software por possuírem capacidades como as de controlar seu próprio comportamento através da autonomia ou de melhorá-lo através do aprendizado são uma excelente abstração para lidar com essa complexidade.

Este trabalho contribui para as áreas de desenvolvimento de agentes de software e de engenharia de software multiagente através da definição de uma arquitetura para o desenvolvimento de agentes de software híbridos com aprendizagem e do desenvolvimento do agente HyLAA para detecção de intrusões em rede de computadores baseado nesta arquitetura. O diferencial desta arquitetura é a aprendizagem de comportamento reativo a partir de comportamento deliberativo frequente. Assim, ao longo do tempo, ações que são frequentemente realizadas de forma deliberativa, vão sendo aprendidas e passam a serem executadas de forma reativa e, portanto, mais rapidamente. Essa abordagem permite que o agente se adapte ao seu ambiente ao longo do tempo e melhore tanto a sua efetividade, ou seja, a sua capacidade de agir corretamente, quanto o seu tempo de resposta, capacidade de realizar ações de forma mais rápida. Neste trabalho, também foi realizada a generalização da arquitetura específica HyLAA que utiliza raciocínio baseado em casos e aprendizagem supervisionada para a arquitetura de referência HyLARA que permite o desenvolvimento de agentes utilizando diferentes tipos de raciocínio e técnicas de aprendizagem.

O agente HyLAA foi avaliado quanto a sua efetividade, desempenho, esforço de desenvolvimento e adaptabilidade através do desenvolvimento de quatro estudos de casos. No primeiro estudo de caso, avaliou-se a efetividade do componente RBC do agente. No segundo estudo de caso, avaliou-se a acurácia do componente de aprendizagem ao aprender novas regras reativas. No terceiro estudo de caso, avaliou-se a efetividade do agente combinando os componentes reativos e deliberativos com aprendizagem. No quarto estudo de caso, avaliou-se a efetividade e desempenho do agente híbrido, mas sem aprendizagem.

Demonstrou-se a hipótese de pesquisa deste trabalho a partir dos resultados dos quatro estudos de casos realizados, pois foi obtida uma maior efetividade e menor tempo de resposta, crescente ao longo da vida do agente, ao se usar o agente híbrido com aprendizagem para detectar intrusões do que o agente híbrido sem aprendizagem ou agente deliberativo ou reativo isoladamente.

6.1 Principais contribuições

O desenvolvimento de agentes de software efetivos e com bom desempenho ainda é um desafio na construção de sistemas multiagentes. Neste trabalho, os objetivos da tese foram alcançados através da especificação da arquitetura híbrida com aprendizagem HyLAA para o desenvolvimento de agentes de software, sua avaliação através do desenvolvimento de um estudo de caso na área da Segurança da Informação, e da generalização desta arquitetura para a arquitetura de referência HyLARA. Outras contribuições do trabalho são a análise e sistematização do conhecimento acerca do desenvolvimento de agentes de software, especialmente dos agentes de software híbridos e das arquiteturas do estado da arte para o desenvolvimento de agentes de software híbridos.

A arquitetura HyLAA, desenvolvida neste trabalho, tem por diferencial a aprendizagem de comportamento reativo a partir de comportamento deliberativo frequente. Esta abordagem traz duas vantagens principais. A primeira é que ela permite que o agente se adapte as mudanças do ambiente de forma automática, através da aprendizagem, diminuindo a necessidade de manutenção evolutiva que, normalmente, tem um custo alto e requerer muito esforço do desenvolvedor. A segunda é que ao aprender comportamento reativo, a efetividade e o desempenho do comportamento do agente melhora ao longo do tempo como foi demonstrado nos estudos de casos realizados.

A efetividade e o desempenho da arquitetura HyLAA foram avaliadas através do desenvolvimento de quatro estudos de casos. Os resultados dos três primeiros estudos de casos mostraram que o comportamento híbrido tem melhor efetividade do que os comportamentos deliberativo ou reativo e um melhor desempenho do que o comportamento deliberativo. Já com os resultados do quarto estudo de caso confirmou-se que, na medida em que o agente aprende, a sua efetividade melhora. Desta forma, os resultados confirmaram que, ao se utilizar a arquitetura HyLARA para o desenvolvimento de arquiteturas concretas de agente híbridos, tem-se uma melhora na efetividade e o desempenho do agente de software através da aprendizagem.

Os estudos de casos experimentais realizados foram adequados para a avaliação da arquitetura HyLARA na construção de uma aplicação concreta. Considera-se como vantagens da arquitetura HyLARA, o aumento da efetividade e do tempo de resposta do agente híbrido desenvolvido, sua adaptabilidade ao ambiente e a diminuição do esforço de desenvolvimento de agentes de software, uma vez que os seus componentes e relacionamentos são bem-definidos.

Outro ponto forte é a aprendizagem automática de regras reativas através da aprendizagem supervisionada, evitando com isso o esforço da sua construção manual.

A HyLARA, especificada a partir da generalização da arquitetura HyLAA, é uma arquitetura de referência e, por tanto, possui um alto nível de abstração permitindo o desenvolvimento de agentes em diversos tipos de domínio, técnicas de aprendizagem e diferentes mecanismos de raciocínio. Além disso, através do mapeamento da arquitetura de referência para uma arquitetura concreta, diminuiu-se o esforço para criação das arquiteturas específicas através do reuso das técnicas de raciocínio e aprendizagem.

Outra contribuição foi o desenvolvimento e utilização da ontologia ONTOID na construção do agente HyLAA. As ontologias enquanto estruturas de representação do conhecimento adotadas pela arquitetura HyLARA tem vantagens em relação a outras estruturas de representação, como a representação semântica do conhecimento, melhorando a efetividade na recuperação de informação e facilitando o seu reuso e integração com outras ontologias. A ONTOID, por ser uma ontologia de aplicação que representa tanto conceitos de domínio quanto de tarefas, pode ser facilmente reusada para construção de outras ontologias de aplicação na área da segurança da informação e utilizada em sistemas baseados em conhecimento nesse domínio.

6.2 Limitações da atual tese

A arquitetura HyLARA foi avaliada em um único domínio de aplicação, utilizando um tipo de raciocínio e uma técnica de aprendizagem específicos (raciocínio RBC e aprendizagem supervisionada, respectivamente). No entanto, ela é uma arquitetura de referência que será utilizada para construir agentes de software híbridos implementados com diferentes tipos de raciocínio e técnicas de aprendizagem, assim é importante explorá-la com diferentes configurações para melhorar a sua aplicabilidade. Portanto, considera-se o uso da arquitetura de referência na construção de uma única aplicação concreta como uma limitação atual da arquitetura de referência tese. Todavia, ela constitui um passo importante para o incentivo às novas produções acadêmicas que vem sendo desenvolvidas no GESEC. Outra limitação da versão atual da arquitetura é que a mesma contempla apenas o projeto detalhado do agente, não contemplando questões como a cooperação e coordenação no contexto de sistemas multiagente, o que pode ser ampliado através de novos estudos e novas aplicações da arquitetura.

6.3 Trabalhos futuros

No contexto do grupo GESEC, pretende-se realizar alguns trabalhos relacionados a arquitetura HyLARA. Outros estudos de casos deverão ser realizados variando tanto as técnicas de aprendizagem quanto o mecanismo de raciocínio do agente. Um destes trabalhos é o do aluno de doutorado Geovane Bezerra da Silva Junior que está reusando a arquitetura híbrida HyLARA para o desenvolvimento de um agente para indução automática de argumentos jurídicos utilizando para isso aprendizagem de máquina supervisionada. O agente desenvolvido pelo aluno será avaliado no domínio das licitações públicas e do direito previdenciário. O objetivo é que estes trabalhos contribuam para a avaliação e maturidade da arquitetura HyLARA.

Outro trabalho a ser realização é a inclusão das diretrizes para construção de arquiteturas de agentes de software híbridos com aprendizagem, segundo a arquitetura de referência HyLARA, no processo MADAE-Pro, já abordado no Capítulo 3, e no ambiente MADAE-IDE que suporta parcialmente o desenvolvimento de agentes de software segundo o processo MADAE-Pro. Esta incorporação é importante, pois tanto o MADAE-Pro quanto o MADAE-IDE contemplam todo o ciclo de desenvolvimento de software, mas não suportam o desenvolvimento de arquiteturas de software híbridas.

Uma extensão da arquitetura de referência HyLARA para contemplar a fase de projeto arquitetural de um agente de software é prevista também como trabalho futuro, uma vez que as habilidades de cooperação e coordenação são muito relevantes em alguns domínios de aplicação.

Na especificação da arquitetura HyLARA atual, abordamos a técnica de aprendizagem baseada em árvores de decisão [46]. Usando esta técnica, o comportamento deliberativo frequente do agente foi sendo transformado em reativo, melhorando a sua efetividade e desempenho ao longo do tempo. Outra abordagem de aprendizagem que será explorada futuramente é a aprendizagem de regras de inferência associadas a um determinado domínio de aplicação. A aprendizagem destas regras poderia ser usada pelo componente deliberativo, potencialmente melhorando a sua efetividade.

6.4 Publicações

As publicações realizadas que incluem os avanços e resultados obtidos ao longo do desenvolvimento deste trabalho foram organizadas em artigos publicados em periódicos, capítulos de livro e artigos publicados em anais de conferência.

6.4.1 Publicações no tópico central da tese

6.4.1.1 Artigos em Periódicos

- Leite, A.; Girardi, R. A Hybrid and Learning Agent Architecture for Network Intrusion Detection. Journal of Systems and Software. 2017. Qualis Capes A2. Fator de impacto: 2.424.

Este artigo apresenta a arquitetura HyLAA utilizada no desenvolvimento de um agente híbrido RBC e os resultados da avaliação realizada, isto é, uma versão condensada deste trabalho.

- Leite, A.; Girardi, R. A Reference Architecture for the Development of Hybrid Learning Software Agents. Journal of Autonomous Agents and Multi-Agent Systems. Qualis Capes A2. Fator de impacto: 1.417. (Artigo Submetido, em processo de avaliação)

Este artigo apresenta a arquitetura de referência HyLARA, generalizada a partir da experiência no desenvolvimento da arquitetura HyLAA.

- Girardi, R. Leite, A. A Survey on Software Agent Architectures, December 2013 Vol. 14 No.1 IEEE Intelligent Informatics Bulletin, Ed. IEEE Intelligent Informatics Bulletin, pp. 8-20. Qualis Capes B4.

Este artigo descreve, exemplifica e discute as principais arquiteturas para desenvolvimento de agente de software atuais e descreve os componentes básicos de um agente de software. Ele está inserido no contexto da primeira etapa da pesquisa.

6.4.1.2 Capítulo de livro

- GIRARDI, R., LEITE, A. A Semantic Approach for Multi-Agent System Design. In: Saqib Saeed, Imran Sarwar Bajwa and Zaigham Mahmood. (Org.). Human Factors in Software Development and Design. 1ed. Hershey: IGI Global, 2014, v., p. 192-218, (capítulo de livro). Sem Classificação Qualis.

Este capítulo de livro reúne conceitos essenciais do projeto de um agente de software e de sistemas multiagente, de reuso de software e de engenharia do conhecimento. Ele está inserido no contexto da primeira etapa da pesquisa (fundamentação teórica).

6.4.1.3 Artigos em eventos internacionais

- GIRARDI, R., LEITE, A. A Reference Architecture of a Hybrid Learning Agent, In 2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI'16). Qualis CAPES: A2.

Este artigo apresenta um resumo da arquitetura de referência híbrida e o seu mapeamento para uma arquitetura especifica.

- LEITE, A., GIRARDI, R. A Case-Based Reasoning Architecture of an Hybrid Software Agent, In Proceedings of the 2014 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2014). Qualis Capes B1.

Este artigo apresenta a arquitetura híbrida utilizando raciocínio baseado em casos aplicados ao direito de família brasileiro. Esse artigo está inserido no contexto da segunda etapa da pesquisa (especificação da solução).

- LEITE, A; GIRARDI, R; NOVAIS, P. Using Ontologies in Hybrid Software Agent Architectures. In: The Knowledge Discovery in Ontologies Workshop at the 2013 IEEE / WIC / ACM International Conference on Web Intelligence (WI13) and Intelligent Agent Technology (IAT-13). Los Alamitos: IEEE Computer Society, Atlanta, 2013. Qualis Capes B1.

Este artigo apresenta a primeira versão da arquitetura híbrida contendo apenas a ideia básica da arquitetura, sem utilizar técnicas de raciocínio ou de aprendizagem e faz um comparativo com outras arquiteturas do estado da arte.

- LEITE, A; GIRARDI, Rosario; NOVAIS, P. Arquitetura Dirigida por Ontologias de um Agente de Software Híbrido. In: 8th Iberian Conference on Information Systems and Technologies, 2013, Lisboa. Proceedings of the 8th Iberian Conference on Information Systems and Technologies, 2013. v. II. p. 338-341. Qualis Capes B4.

Este artigo contém a tese do trabalho de tese submetido a um Simpósio Doutoral contendo a problemática e relevância do mesmo, a descrição inicial da arquitetura, a metodologia de pesquisa e os resultados esperados. Ele também está inserido no contexto da segunda etapa da pesquisa.

6.4.2 Publicações relacionadas com o tópico da tese

6.4.2.1 Artigos em eventos internacionais

- MENESES, R; LEITE, A. GIRARDI, R. Ontologia de Aplicação para o Desenvolvimento de Sistemas de Detecção de Intrusão Baseado em Casos. In: 10ª Conferencia Ibérica de Sistemas y Tecnologías de Información, 2015, Aveiro. Qualis Capes B4.

Este artigo apresenta a ONTOID, utilizada como base de conhecimento do agente. Ele está inserido no contexto da segunda etapa da pesquisa (especificação da solução).

- MENDES, W. C.; GIRARDI, R., LEITE, A. Arquitetura Baseada em Ontologias de um Agente RBC. In: 8th Iberian Conference on Information Systems and Technologies, 2013, Lisboa. Proceedings of the 8th Iberian Conference on Information Systems and Technologies, 2013. V. I. p. 776-781. Qualis Capes B4.

Este artigo apresenta conceitos teóricos acerca do raciocínio baseado em casos, ontologias e uma arquitetura para o desenvolvimento de agentes de software utilizando raciocínio baseado em casos. Ele está inserido no contexto da primeira e segunda etapa da pesquisa.

RFERÊNCIAS BIBLIOGRÁFICAS

[1] AAMODT, A. PLAZA, E. Case-Based Reasoning: Foundation Issues, Methodological Variations, and System Approaches", AICOM, Vol. 7: 1, p. 39-59, 1994.

[2] ABNT. ABNT NBR ISO/IEC 27002 - Código de prática para a gestão de Segurança da informação. Associação Brasileira de Normas Técnicas, Rio de Janeiro, 2005.

[3] ALLAN, J. G. SALTON, BUCKLEY C., Approaches to Passage Retrieval in Full Text Information Systems, TR93-1334, Department of Computer Science, Cornell University, March 1993.

[4] ARORA, S. BAWA, R. STUDENT, M. A Review on Intrusion Detection System to Protect Cloud Data, International Journal of Innovations and Advancement in Computer Science, vol. 3, pp.30–34, 2014.

[5] BALL, J. Explorations in ACT-R Based Cognitive Modeling–Chunks, Inheritance, Production Matching and Memory in Language Analysis. In: Proceedings of the AAAI Fall Symposium: Advances in Cognitive Systems. 2011. pp. 10-17.

[6] BENMOUSSA, H. et al., Towards a new intelligent generation of intrusion detection systems, Proceedings of the 4th Edition of National Security Days (JNS4), pp.1–5, 2014.

[7] BHARGAVA, N. Decision tree analysis on j48 algorithm for data mining. Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering, v. 3, n. 6, 2013.

[8] BISHOP, C. Pattern Recognition and Machine Learning, Springer, 2006.

[9] BSI, ISO/IEC 27001:2005 Information technology — Security techniques — Information security management systems — Requirements. October, vol. 3, p. 44, 2009.

[10] CAVALCANTE, U. GIRARDI, R. An Overview of the MADAE-IDE Multi-Agent System Development Environment. Proceedings of the 7th International Conference on Information Technology: New Generations. Las Vegas. IEEE, 2010. pp. 968-973.

[11] CAVALCANTE, U. MADAE-IDE: Um ambiente de desenvolvimento de software baseado no conhecimento para o reuso composicional no desenvolvimento de sistemas multiagente, Dissertação de mestrado, 2009.

[12] CERNUZZI, L., MOLESINI, A., OMICINI, A., The Gaia Methodology Process. In: Handbook on Agent-Oriented Design Processes. Springer Berlin Heidelberg, 2014. p. 141-172.

[13] CLOUTIER, R., MULLER, G., VERMA, D., NILCHIANI, R., HOLE, E., BONE, M. The Concept of Reference Architectures. Systems Engineering, 13(1), 2010, 14-27.

[14] COPPIN, B. Inteligência artificial. Rio de Janeiro: LTC, 2010.

[15] COSSENTINO, M. SEIDITA, V. PASSI: Process for agent society's specification and implementation. In: Handbook on Agent-Oriented Design Processes. Springer Berlin Heidelberg, 2014. p. 287-329.

[16] DARPA Intrusion Detection Evaluation. Retrieved September 01, 2017, from https://ll.mit.edu/ideval/docs/detections_1999.html

[17] GESEC. Grupo de Engenharia de Software e Engenharia do Conhecimento. Disponível em: http://gesec.deinf.ufma.br/. Acessado em: 04 de novembro de 2016.

[18] GIRARDI, R., LEITE, A. Knowledge Engineering Support for Agent-Oriented Software Reuse. In: M. Ramachandran. (Org.). Knowledge Engineering for Software Development Life Cycles: Support Technologies and Applications. Hershey: IGI Global, v. I, p. 177-195, 2011.

[19] GIRARDI, R.; LEITE, A. The Specification of Requirements in the MADAE-Pro Software Process. iSys: Revista Brasileira de Sistemas de Informação, v. 3, p. 3, 2010.

[20] GIRARDI, Rosario. Guiding Ontology Learning and Population by Knowledge System Goals. In: KEOD. 2010. p. 480-484.

[21] GORDON, M; KOCHEN, M. "Recall"-"Precision" trade-off: A derivation. Ann Arbor, v. 1001, p. 48109, 1989.

[22] GRUBER, T. Toward principles for the design of ontologies used for knowledge sharing, International Journal of Human-Computer Studies, v. 43, n. 5, p. 907-928, 1995.

[23] GUARINO, N., 1998. Formal Ontology in Information Systems: Proceedings of the First International Conference (FOIS'98) 1st ed., Trento, Italy: IOS press.

[24] HALL, M., FRANK, E., HOLMES, G., PFAHRINGER, B., REUTEMANN, P., WITTEN, I. H. The WEKA data mining software: an update. ACM SIGKDD explorations newsletter, 11(1), 10-18, 2009.

[25] HASTIE, T., TIBSHIRANI, R., FRIEDMAN, Jerome. Overview of supervised learning. In: The elements of statistical learning. Springer New York, 2009. p. 9-41.

[26] HMIDA, F. B., CHAARI, W. L., TAGINA, M. (2008, March). Performance evaluation of multiagent systems: communication criterion. In KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications (pp. 773-782). Springer Berlin Heidelberg.

[27] JAIN, J., PAL, P. A Recent Study over Cyber Security and its Elements. International Journal, v. 8, n. 3, 2017.

[28] JAIN, V., SINGH, M. Ontology Development and Query Retrieval using Tool. International Journal of Intelligent Systems and Applications (IJISA), v. 5, n. 9, p. 67, 2013.

[29] JENNINGS, NICHOLAS R., SYCARA, K., WOOLDRIDGE, M. A roadmap of agent research and development. Autonomous agents and multi-agent systems, v. 1, n. 1, p. 7-38, 1998.

[30] KAELBLING, L., LITTMAN, Michael L., MOORE, A. Reinforcement learning: A survey. Journal of artificial intelligence research, v. 4, p. 237-285, 1996.

[31] KENDALL, E. A. KRISHNA, P. V. PATHAK, C.V. SURESH, C. B. Patterns of intelligent and mobile agents. In Proceedings of the Second International Conference on Autonomous Agents, 1998, pp. 92-99.

[32] KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. Ijcai. Vol. 14. No. 2. 1995.

[33] KOWALSKI, R. A. Prolog as a logic programming language. Department of Computing Imperial College of Science & Technology University of London, 1981.

[34] LAIRD, E. The Soar cognitive architecture. MIT Press, 2012.

[35] LAIRD. J. The Soar 9 Tutorial, University of Michigan, 2012.

[36] LANGLEY, P., LAIRD, J., ROGERS, S. Cognitive architectures: Research issues and challenges. Cognitive Systems Research, v. 10, n. 2, p. 141-160, 2009.

[37] LEITE, A., GIRARDI, A. ONTORMAS: Uma ferramenta dirigida por ontologias para a Engenharia de Domínio e de Aplicações Multiagente, The XI Iberoamerican Workshop on Requirements Engineering and Software Environment, Pernambuco, 2008.

[38] LEITE, A., GIRARDI, R. A Process for Multi-Agent Domain and Application Engineering: The Domain Analysis and Application Requirements Engineering Phases, Proceedings of the 11th International Conference on Enterprise Information Systems, Ed. INSTIIC. Milan, Italy, 2009.

[39] LICATO, J., SUN, R., BRINGSJORD, S. Using a Hybrid Cognitive Architecture to Model Children Errors in an Analogy Task. In: Proceedings of CogSci. 2014.

[40] LLOYD, L.W., Legal Reason: The Use of Analogy in Legal Argument. New York: Cambridge University Press, 2005, p.192.

[41] LPA WIN PROLOG. Disponível em: http://www.lpa.co.uk/win.htm. Acessado em: 27/09/2013.

[42] MARKOV, Z. Induction of Decision Trees Algorithm. Disponível em: http://www.cs.ccsu.edu/~markov/ccsu_courses/MachineLearning.html. Acesso em 31/07/2015.

[43] MENESES, R. LEITE, A., GIRARDI, R. Ontologia de Aplicação para o Desenvolvimento de Sistemas de Detecção de Intrusão Baseado em Casos. In Information Systems and Technologies (CISTI), 2015 10th Iberian Conference on. IEEE, 2015. p. 1-4.

[44] MENESES, R. Uma Ontologia da Aplicação para Apoio à Tomada de Decisões em Situações de Ameaças à Segurança da Informação, Dissertação de Mestrado, Universidade Federal do Maranhão, 2015.

[45] MINSKY, M. A framework for representing knowledge. In: Computation & intelligence. American Association for Artificial Intelligence, 1995, p. 163-189.

[46] MITCHELL, T. Machine Learning, McGraw-Hill, Book, USA, 1997.

[47] MOLYNEAUX, I. The Art of Application Performance Testing: From Strategy to Tools. O'Reilly Media, Inc., 2014.

[48] MOTIK, B. OWL 2 Web Ontology Language: Structural specification and functional-style syntax. W3C recommendation, v. 27, p. 17, 2009.

[49] MÜLLER, P.; PISCHEL, Markus. The agent architecture InteRRaP: Concept and application. 2011.

[50] MUSEN, Mark A. The protégé project: a look back and a look forward. AI matters, v. 1, n. 4, p. 4-12, 2015.

[51] MYLOPOULOS, J., CASTRO, J., KOLP, M. The Evolution of Tropos. In: Seminal Contributions to Information Systems Engineering. Springer Berlin Heidelberg, 2013, p. 281-287.

[52] NSL-KDD Dataset. Retrieved September 01, 2017, from http://www.unb.ca/cic/research/datasets/nsl.html

[53] OLUSOLA, A. OLADELE, S. A., ABOSEDE, D. O. Analysis of KDD'99 Intrusion detection dataset for selection of relevance features. In: Proceedings of the World Congress on Engineering and Computer Science, p. 20-22, 2010.

[54] QINZHOU, N., Lei, H. Design of case-based hybrid agent structure for machine tools of intelligent design system, Software Engineering and Service Science (ICSESS), 2012 IEEE 3rd International Conference on , pp.59-62, 2012.

[55] QUINLAN, J. C4.5: programs for machine learning. Elsevier, 2014.

[56] QUINLAN, J. R. Induction of decision trees. Machine learning, 1(1), 81-106, 1986.

[57] REVATHI, S.; MALATHI, A. A detailed analysis on NSL-KDD dataset using various machine learning techniques for intrusion detection. International Journal of Engineering Research and Technology. ESRSA Publications, 2013.

[58] RIESBECK, K., SCHANK, C. Inside Case-based Reasoning. Psychology Press, 2013.

[59] RUGGIERI, S. Efficient C4.5 [classification algorithm]. IEEE transactions on knowledge and data engineering, v. 14, n. 2, p. 438-444, 2002.

[60] RUSSEL, S., NORVIG, P. Artificial Intelligence: A Modern Approach (3nd ed.), Prentice-Hall, 2009.

[61] SANTOS, L., GIRARDI, R., NOVAIS, P. A case study on the construction of application ontologies. In: Information Technology: New Generations (ITNG), 2013 Tenth International Conference on. IEEE, 2013. p. 619-624.

[62] SANTOS, S., GIRARDI, R., Apponto-Pro: An incremental process for ontology learning and population. In: Information Systems and Technologies (CISTI), 2014 9th Iberian Conference on. IEEE, 2014. p. 1-6.

[63] SAWANT, T. S. ITKAR S. A. A Survey and Comparative Study of Different Data Mining Techniques for Implementation of Intrusion Detection System, International Journal of Current Engineering and Technology, vol. 4, pp.1288–1291, 2014.

[64] SILVA, F., GIRARDI, R. An Approach to Join Ontologies and their Reuse in the Construction of Application Ontologies. In: Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2014 IEEE/WIC/ACM International Joint Conferences on. IEEE, 2014. p. 424-431.

[65] SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. Information Processing & Management, v. 45, n. 4, p. 427-437, 2009.

[66] STERLING, L., SHAPIRO, E. The Art of Prolog: Advanced Programming Techniques, 2nd Edition, MIT Press, 1994, 688 pages, ISBN 0-262-19338-8.

[67] SUN, R. The CLARION cognitive architecture: Extending cognitive modeling to social simulation. Cognition and multi-agent interaction, p. 79-99, 2006.

[68] SUN, Y. Wu, B. Agent Hybrid Architecture and Its Decision Processes, International Conference on Machine Learning and Cybernetics, pp.641-644, 2006.

[69] SUTTON, Richard S.; BARTO, Andrew G. Reinforcement learning: An introduction. Cambridge: MIT press, 1998.

[70] TAATGEN, Niels A.; LEBIERE, Christian; ANDERSON, John R. Modeling paradigms in ACT-R. Cognition and multi-agent interaction: From cognitive modeling to social simulation, p. 29-52, 2006.

[71] THAGARD, P., SHELLEY, E.C., Abductive Reasoning: Logic, Visual Thinking, and Coherence, In: M.L, 1997.

[72] VASSILIADIS, V., WIELEMAKER, J., MUNGALL, C. Processing OWL2 Ontologies using Thea: An Application of Logic Programming. In: OWLED. 2009.

[73] VIVEKANANDAN, K.; RAMA, D. Analysing the Scope for Testing in PASSI Methodology. International Journal, v. 3, n. 1, 2013.

[74] VON ZUBEN, J., ATTUX, R. Árvores de decisão, DCA/FEEC/Unicamp, v.2, 2016.

[75] WANGENHEIM, C.; WANGENHEIM, A. Raciocínio Baseado em Casos, 1. ed. Barueri-SP: Manole, 2003.

[76] WEISS, G., Multiagent Systems (Intelligent Robotics and Autonomous Agents series), The MIT Press, 2013.

[77] WHITMAN, M. E. In defense of the realm: understanding the threats to information security. International Journal of Information Management, v. 24, n. 1, p. 43-57, 2004.

[78] WOOLDRIDGE, M. An Introduction to Multi-agent Systems (2nd ed.), Wiley Publishing, 2009.

[79] WU, X., SUN, J. Study on a KQML-based intelligent multi-agent system. In: Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on. IEEE, 2010. p. 466-469.

APÊNDICE 1 – Documentação da implementação do agente HyLAA

O agente HyLAA foi desenvolvido em Prolog, utilizando o ambiente LPA Prolog e o algoritmo C4.5 implementado em Java. Para realização das experiências, as percepções do agente foram armazenadas em um arquivo denominado percepcoes.pl. O agente tenta usar o conjunto de regras reativas aprendidas disponíveis na ontologia ONTOID (arquivo ontoid.pl) para encontrar uma solução e se não encontrar usa o seu componente deliberativo RBC. Todos os resultados da execução do agente (detecção de intrusão e solução recomendada) são mostrados na tela no momento da execução e gravados em arquivo.

Os predicados principais implementados no agente estão listados abaixo:

% start_ HyLAA _agent /1.

% O agente é inicializado, monitora o ambiente e se tiver uma nova percepção, instancia um novo caso problema, calcula a similaridade com os casos da base de conhecimento (detecta intrusões) e recomenda o caso mais similar (ação do agente).

%argumentos de entrada
% não há

%argumentos de saída.
% se detectar uma intrusão, o agente recomenda uma solução para o mesmo.

%Exemplo de execução
| ?- start_ HyLAA _agent.

Agente monitorando o ambiente
0.684 seconds to consult
f:\adriana\pendrive11042015\pendrive15032015\implementação\agente\ontoid.pl

Abolishing f:\adriana\pendrive11042015\pendrive15032015\implementação\agente\ontoid.pl

0.684 seconds to consult
f:\adriana\pendrive11042015\pendrive15032015\implementação\agente\ontoid.pl

Problema mais similar encontrado: O1#IntrusionProblem_41

Ataque Relacionado: apache2

Similaridade: 0.926827600000001

Solução: Instalar e ativar o o mod_evasive do Apache
yes

%Predicado comparacasos/4

%Compara o novo caso problema com a lista de casos preexistentes na base de conhecimento e retorna uma lista de casos similares. Os casos similares são aqueles que possuírem similaridade igual ou maior que o ponto de corte utilizado.

%comparacasos(NovoCasoProblema,ListaCasosBase,ListaInicial,ListaCasosSimilares)
%argumentos de entrada
% NovoCasoProblema
% ListaCasosBase
% ListaInicial

%argumentos de saída
% ListaCasosSimilares

%Exemplo de execução
%?- comparacasos('O1#MonitoredPackage_2',
['O1#IntrusionProblem_1','O1#IntrusionProblem_2','O1#IntrusionProblem_3','O1#IntrusionProblem_4'],[],ListaCasosSimilares).
%ListaCasosSimilares =
[('O1#IntrusionProblem_4',0.3902432),('O1#IntrusionProblem_3',0.3902432),('O1#IntrusionProblem_2',0.4146334),('O1#IntrusionProblem_1',0.3902432)]

%Predicado recomenda_solução/1

%recomenda a solução para o novo caso problema, mostra o ataque mais similar a ele e a similaridade.

%recomenda_solução(NovoCasoProblema)
%argumentos de entrada
% não há

%argumentos de saída.
% não há, o predicado apenas faz impressões na tela

%Exemplo de execução
%?- recomenda_solução.

%# Abolishing c:\users\adriana\implementação\ontoid.pl
%# 0.047 seconds to consult c:\users\adriana\implementação\ontoid.pl
%Problema mais similar encontrado: O1#IntrusionProblem_4
%Ataque Realacionado: O1#Land
%Similaridade: 0.487804
%Solução: Use the firewall as a relay between the server end its clients.
yes

%calculaSimilaridade/3.

%calculaSimilaridade(NovoCasoProblema,Caso,Similaridade)

%Faz o cálculo de similaridade entre um novo caso problema, ou seja, um novo pacote de
%dados de uma rede de computadores e um caso qualquer da base de dados (ONTOID).

%argumentos de entrada
%	NovoCasoProblema
%	Caso

%argumentos de saída
%	Similaridade

%Exemplo de execução

%?calculaSimilaridade('O1#MonitoredPackage_2','O1#IntrusionProblem_1',Similaridade).
% Similaridade = 0.3902432

%calculaSimilaridadeEntreCasos/2.

%Calcula a similaridade de um novo caso problema com todos os casos da base e retorna a lista de casos similares

%calculaSimilaridadeEntreCasos(NovoCasoProblema,ListaCasosSimilares)

% argumentos de entrada
%	NovoCasoProblema

% argumentos de saída
%	ListaCasosSimilares

% Exemplo de execução

% ?- calculaSimilaridadeEntreCasos(`O1#MonitoredPackage_1`,ListaCasosSimilares).
% ListaCasosSimilares =
[('O1#IntrusionProblem_636',0.7073158),('O1#IntrusionProblem_635',0.7073158),
('O1#IntrusionProblem_634',0.7073158),('O1#IntrusionProblem_631',0.7073158),('O1#Intrusion
Problem_629',0.7073158),
('O1#IntrusionProblem_619',0.7073158),('O1#IntrusionProblem_617',0.7073158),('O1#Intrusion
Problem_611',0.7073158),('O1#IntrusionProblem_608',0.7073158),('O1#IntrusionProblem_607',
0.7073158),('O1#IntrusionProblem_605',0.7073158),
('O1#IntrusionProblem_203',0.7073158),('O1#IntrusionProblem_202',0.731706000000001),('O1
#IntrusionProblem_201',0.7073158),
('O1#IntrusionProblem_200',0.7073158),('O1#IntrusionProblem_197',0.731706000000001),('O1
#IntrusionProblem_195',0.7073158), ('O1#IntrusionProblem_193',0.7073158)

%selecionaCasoMaisSimilar/2.

%Seleciona o caso mais similar da lista de casos similares

%selecionaCasoMaisSimilar(ListaCasosSim,CasoMaisSim)

% argumentos de entrada
% ListaCasosSim

% argumentos de saída
% CasoMaisSim

% Exemplo de execução

% ?-
selecionaCasoMaisSimilar([('O1#IntrusionProblem_636',0.7073158),('O1#IntrusionProblem_63
5',0.7073158),('O1#IntrusionProblem_634',0.7073158)],CasoMaisSim).

% CasoMaisSim = ('O1#IntrusionProblem_636',0.7073158) ;

%calculaPrecisionAndRecall/4.

%Calcula e retorna o valor da precisão e do recall na recuperação de cada lista de casos similares ao NCP

%calculaPrecisionAndRecall(NCP,ListCasesSim,Precision,Recall)

% argumentos de entrada
% NCP
% ListCasesSim

```
%       Weight


% argumentos de saída
%       Precision
%       Recall


% Exemplo de execução

% ?-
calculaPrecisionAndRecall(`O1#MonitoredPackage_1`,[('O1#AttackProblem_636',0.7073158),('
O1#AttackProblem_635',0.7073158),
('O1#AttackProblem_634',0.7073158),('O1#AttackProblem_631',0.7073158),('O1#AttackProble
m_629',0.7073158),
('O1#AttackProblem_619',0.7073158),('O1#AttackProblem_617',0.7073158),('O1#AttackProble
m_611',0.7073158),('O1#AttackProblem_608',0.7073158),('O1#AttackProblem_607',0.7073158)
,('O1#AttackProblem_605',0.7073158),
('O1#AttackProblem_203',0.7073158),('O1#AttackProblem_202',0.731706000000001)],Precisio
n,Recall).
O1#MonitoredPackage_1
AttackNCP: apache2

AmountRCB: 737O1#AttackProblem_636,apache2
O1#AttackProblem_635,apache2
O1#AttackProblem_634,apache2
O1#AttackProblem_631,apache2
O1#AttackProblem_629,apache2
O1#AttackProblem_619,apache2
O1#AttackProblem_617,apache2
O1#AttackProblem_611,apache2
O1#AttackProblem_608,apache2
O1#AttackProblem_607,apache2
O1#AttackProblem_605,apache2
O1#AttackProblem_203,apache2
O1#AttackProblem_202,apache2
Precision = 1 ,
Recall = 0.0176391 ;

no
```

%instancia/2.

```
%cria uma instancia de um pacote de dados na base de conheciemnto.

%instancia(PacoteDeDados, NovoCasoProblema).

% argumentos de entrada
```

%	PacoteDeDados

% argumentos de saída
%	NovoCasoProblema

% Exemplo de execução
% ?-instancia
([0,tcp,private,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,229,10,0.00,0.00,1.00,1.00,0.04,0.06,0.00,255,10,0.04,0.06,0.00,0.00%,0.00,0.00,1.00,1.00], NovoCasoProblema).

%NovoCasoProblema = 'MonitoredPackage_1'

%OBS: na base de conheciemento é gerado todos os atributos do novo caso problema com seus respectivos valores.

% lerPacote/1.

%ler, linha por linha, os pacotes de dados de uma rede de computadores a partir de um %arquivo contendo esses pacotes e retorna uma lista contendo os dados de cada pacote". %Obs: cada vez que esse predicado for executado, enquanto o programa está em execução, %ele pega a proxima linha do arquivo.

%lerPacote(ListaDados)

% argumentos de entrada
%	Não há

% argumentos de saída
%	ListaDados.

% Exemplo de execução

%?-lerPacote(ListaDados).
%ListaDados =
[`1`,`0.00`,`2425`,`26`,`0.12`,`0.12`,`0.04`,`0.38`,`0.00`,`10`,`0.00`,`0.30`,`0.00`,`13`,`SF`,`0`,`0`,`0`,`0`,`1`,`0`,`0`,`0`,`0`,`0`,`0`,`0`,`tcp`,`0.00`,`0`,`1.00`,`0.00`,`telnet`,`118`,`1`,`0.00`,`0.00`,`0.00`,`0`,`0`,`0`]

%insereCasoSimilar/4

%insere um caso similar com o seu valor de similaridade em uma lista no formato de par ordenado(Caso, Similaridade)

%insereCasoSimilar(Caso,Similaridade,ListaInicial,ListaCasoSim)
%argumentos de entrada
%	NovoCasoProblema
%	Similaridade
%	ListaInicial

%argumentos de saída
%	ListaCasoSim

%Exemplo de execução

%?- insereCasoSimilar('O1#AttackProblem_2',0.89,[],ListaCasoSim).
%ListaCasoSim = [('O1#AttackProblem_2',0.89)]

%countRelevantsRetrivedCases/4.

%Conta e retorna a quantidade de casos relevantes ao NCP, há na base de conhecimento

%countRelevantsRetrivedCases(ListCasesSim,Attack,AmountRRCEntrada,AmountRRC)

% argumentos de entrada
%	ListCasesSim
%	Attack
%	AmountRRCEntrada

% argumentos de saída
%	AmountRRC

% Exemplo de execução

% ?-
countRelevantsRetrivedCases([('O1#AttackProblem_636',0.7073158),('O1#AttackProblem_635',0.70731
58),
('O1#AttackProblem_634',0.7073158),('O1#AttackProblem_631',0.7073158),('O1#AttackProblem_629',0
.7073158),
('O1#AttackProblem_619',0.7073158),('O1#AttackProblem_617',0.7073158),('O1#AttackProblem_611',0
.7073158),('O1#AttackProblem_608',0.7073158),('O1#AttackProblem_607',0.7073158),('O1#AttackProbl
em_605',0.7073158),
('O1#AttackProblem_203',0.7073158),('O1#AttackProblem_202',0.731706000000001)],`apache2`,0,Amo
untRRC).
O1#AttackProblem_636,apache2
O1#AttackProblem_635,apache2
O1#AttackProblem_634,apache2
O1#AttackProblem_631,apache2
O1#AttackProblem_629,apache2
O1#AttackProblem_619,apache2
O1#AttackProblem_617,apache2
O1#AttackProblem_611,apache2
O1#AttackProblem_608,apache2
O1#AttackProblem_607,apache2
O1#AttackProblem_605,apache2
O1#AttackProblem_203,apache2
O1#AttackProblem_202,apache2
AmountRRC = 13 ;

APÊNDICE 2 – Tutorial de execução do agente HyLAA para reprodução dos estudos de casos realizados no capítulo de avaliação

Para reprodução do primeiro estudo de caso, siga os seguintes passos:

1- Compile o arquivo “agenteHyLAA.pl” utilizando o software LPA-Prolog.
2- No ambiente de excução do LPA-Prolog, digite o seguinte comando:
% ?- start_ HyLAA _agent.
A única função deste predicado é chamar três outros predicados: **memoria**, **inicializa e recomenda_solução**, que também podem ser executados separadamente.

O comando **memoria**, que pode ser executado como abaixo:

% ?- memoria.

Serve para alocar mais memória de trabalho, pois o espaço padrão alocado pelo software é muito reduzido ontologia utilizada (ONTOID) possui muitas instancias (11 mil) o que sobrecarrega o software e causa travamentos se o espaço de memória não for aumentado.

3- Já o comando **inicializa**, que pode ser executado como abaixo:

% ?- inicializa.

Tem por função inicializar o agente e carregar a ontologia ONTOID (Base de casos de intrusão) na memória de trabalho.

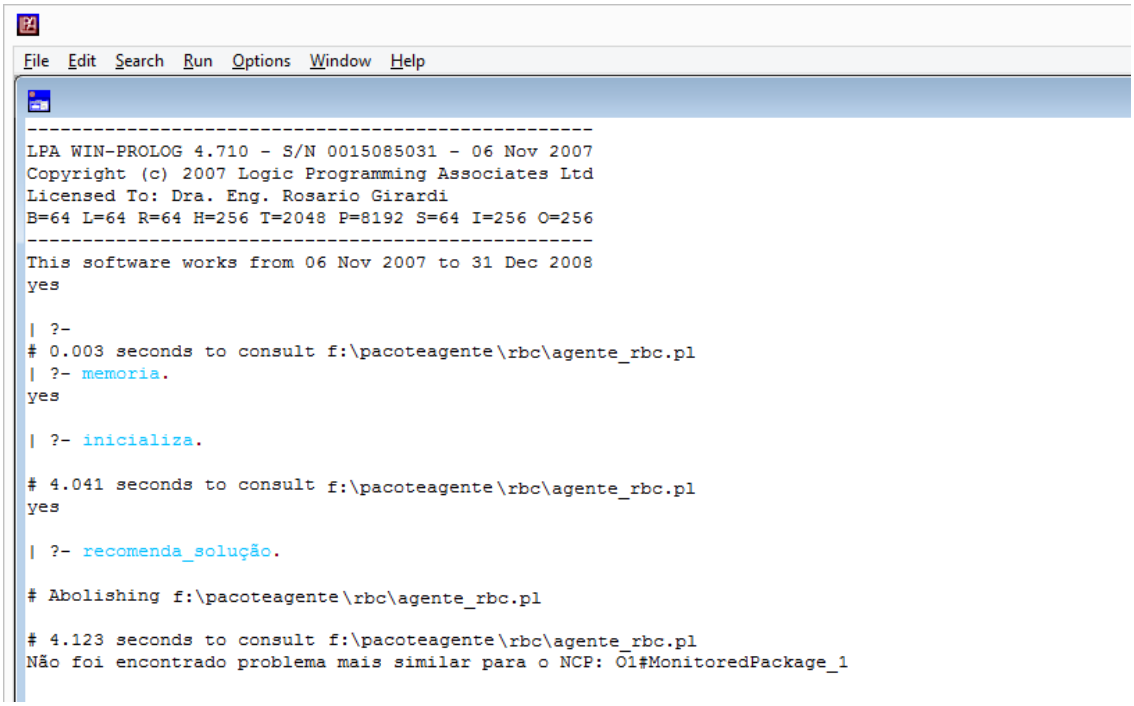
4- Por sua vez, o comando **recomenda_solução**, que pode ser executado como abaixo:

recomenda_solução.

Funciona da seguinte forma, ele lê uma percepção do arquivo percepcoes.pl, detecta o tipo de intrusão e recomenda uma solução.

4- Após a execução do agente, dentro do mesmo diretório vão ser criados os arquivos precisionrecall.pl e resultados.pl, esses arquivos contém os valores de recall e precision das intrusões detectadas, são usados para análise dos experimentos e geração de gráficos.

Na figura a seguir temos um exemplo da execução do agente, onde não foi encontrado na base de conhecimento nenhum caso similar ao novo caso problema (arquivo percepçoes.pl):



APÊNDICE 3 – Exemplos de execução do agente HyLAA para reprodução dos estudos de casos realizados no capítulo de avaliação

Execução do agente para o Estudo de Caso 1: avaliação do componente RBC

Um exemplo da execução das experiências estudo de caso 1 está representada abaixo. A diferença no processamento das duas experiências deste estudo de caso se dá pela atribuição de pesos aos atributos mais relevantes dos casos de intrusão da base de conhecimento na segunda experiência, no entanto, a sua execução é idêntica.

Percepção (um conjunto de dados referentes a uma sessão de comunicação):
`2,0.00,0,231,0.01,0.03,0.00,0.99,0.00,228,0.00,0.03,0.00,748,RSTR,0,0,0,0,1,0,0,0,0,0,0,0,tcp,1.00,0,1.00,0.00,http,91544,2,0.00,1.00,0.00,0,0,0,#`.

Ação: A ação do agente consiste em identificar o tipo de intrusão associada à percepção acima, neste exemplo uma intrusão do tipo Smurf (linha 7), informar o valor de similaridade entre a percepção e o caso mais similar, neste exemplo o valor de similaridade foi de 90% (linha 8) e recomendar uma solução para cessar ou prevenir essa intrusão (linha 9) :

1. |?-start_ HyLAA _agent.
2. * HyLAA Hybrid Agent waiting new perceptions…*
3. # Abolishing f:\HyLAA_Agent\ ontoid.pl
4. # 0.969 seconds to consult f:\HyLAA_Agent\ontoid.pl
5. New Case Problem: MonitoredPackage_1
6. Most similar problem: O1#IntrusionProblem_1
7. **Intrusion type: Smurf**
8. **Similarity value: 0.90**
9. **Recommended solution: *Block directed broadcast traffic coming into network**

Execução do agente para o Estudo de Caso 2: avaliação do componente de aprendizagem

Um exemplo da execução das experiências do estudo de caso 2 está representado abaixo. A diferença no processamento das três experiências se dá apenas pelo tamanho do conjunto de treinamento utilizado para gerar as regras aprendidas, no entanto, a sua execução é idêntica.

Percepção (um conjunto de dados referentes a uma sessão de comunicação):
`13,0.00,0,255,0.01,0.20,0.00,0.93,0.00,237,0.00,0.22,0.00,2088,RSTR,0,0,0,0,1,0,0,0,0,0,0,0,tcp,0.92,0,1.00,0.08,http,72564,13,0.00,0.92,0.08,0,0,0,#`.

Ação: A ação do agente consiste em identificar a intrusão associada a percepção acima, neste exemplo a intrusão detectada foi do tipo PoD (linha 6) utilizando o conjunto de regras reativas aprendidas e recomendar uma solução para cessar ou prevenir essa intrusão, como neste exemplo mostra (linha7) :

```
1. |?-start_ HyLAA_agent.
2. * HyLAA Hybrid Agent waiting new perceptions…*
3. # Abolishing f:\HyLAA_Agent\ ontoid.pl
4. # 0.969 seconds to consult f:\HyLAA_Agent\ontoid.pl
5. New Case Problem: MonitoredPackage_2
6. Intrusion type: PoD
7. Recommended solution (reactive): *Configure individual hosts and routers to not respond to ICMP requests or broadcast
```

Execução do agente para o Estudo de Caso 3: avaliação do agente híbrido com aprendizagem

Um exemplo da execução da experiência do estudo de caso 3 está representado abaixo:

Percepção (um conjunto de dados referentes a uma sessão de comunicação):
`272,0.06,0,255,0.06,1.00,0.00,0.03,0.00,8,0.00,1.00,0.00,0,REJ,0,0,0,0,0,0,0,0,0,0,0,0,tcp,1.00,0,0.03,0.00,uucp_path,0,8,0.00,1.00,0.00,0,0,0,#`.

Ação: A ação do agente consiste em identificar a intrusão associada a percepção acima (linha 6) utilizando o conjunto de regras reativas aprendidas (quando disponíveis) ou utilizar o RBC e recomendar uma solução para cessar ou prevenir essa intrusão (linha7) :

```
1. |?-start_ HyLAA_agent.
2. * HyLAA Hybrid Agent waiting new perceptions…*
3. # Abolishing f:\HyLAA_Agent\ ontoid.pl
4. # 0.969 seconds to consult f:\HyLAA_Agent\ontoid.pl
5. New Case Problem: MonitoredPackage_2
6. Intrusion type: Neptune
7. Recommended solution (reactive): *Configure individual hosts and routers to not respond to ICMP requests or broadcast
```

Percepção de realimentação: A percepção de realimentação, consiste em o agente perceber o resultado da execução de uma ação no ambiente. Ela é composta da percepção, ação e resultado. Neste estudo de caso, o resultado da execução de uma ação no ambiente foram definidos explicitamente pelo usuário como “Successful” ou “Fail” baseado no conjunto de dados NSL-KDD que continha além da percepção correspondente a uma intrusão, a sua solução, ou seja, o tipo de intrusão associado. Dessa forma, quando o agente acertava qual o tipo de intrusão de uma determinada percepção, era acrescentado ao conjunto <percpeção, ação> o resultado <Successful> e quando ele errava era acrescentado o resultado <“Fail”> a esse conjunto, como representado nos exemplos a seguir.

`13,0.00,0,255,0.01,0.20,0.00,0.93,0.00,237,0.00,0.22,0.00,2088,RSTR,0,0,0,0,1,0,0,0,0,0,0,0,tcp,0.92,0,1.00,0.08,http,72564,13,0.00,0.92,0.08,0,0,0,Neptune# ,Sucessful`. => Recomendação correta

`13,0.00,0,255,0.01,0.20,0.00,0.93,0.00,237,0.00,0.22,0.00,2088,RSTR,0,0,0,0,1,0,0,0,0,0,0,0,tcp,0.92,0,1.00,0.08,http,72564,13,0.00,0.92,0.08,0,0,0, PoD#, Fail`. = Recomendação incorreta

Padrão de desempenho: Como se utilizou uma forma explícita, informada pelo usuário, para se obter o resultado da ação do ambiente o padrão de desempenho (regras utilizadas pelo Crítico) ficaram bastante simplificadas como ilustradas no exemplo abaixo:

```
if (percepcao = X) and (acao = Y) and (resultadoacao= “Sucessful”) then
store example(percepcao, acao)
```

Assim, dos exemplos anteriores (Neptune e Pod), apenas o exemplo abaixo (Neptune) seria armazenado para treinamento.

Exemplo incluído no conjunto de treinamento:

`13,0.00,0,255,0.01,0.20,0.00,0.93,0.00,237,0.00,0.22,0.00,2088,RSTR,0,0,0,0,1,0,0,0,0,0,0,0,tcp,0.92,0,1.00,0.08,http,72564,13,0.00,0.92,0.08,0,0,0, Neptune# `.

Execução do agente para o Estudo de Caso 4: avaliação do agente híbrido sem aprendizagem

Um exemplo da execução das experiências 1 e 2 do estudo de caso 4 está representada abaixo, as duas experiências diferem apenas quanto ao conjunto utilizado como percepção do agente:

Percepção (um conjunto de dados referentes a uma sessão de comunicação):
`102,0.07,0,255,0.06,0.00,0.00,0.01,1.00,2,0.00,0.00,1.00,0,S0,0,0,0,0,0,0,0,0,0,0,0,tcp,0.00,0,0.02,1.00,ssh,0,2,0.00,0.00,1.00,0,0,0,#`.

Ação: A ação do agente consiste em identificar a intrusão associada a percepção acima (linha 6) utilizando o conjunto de regras reativas aprendidas e recomendar uma solução para cessar ou prevenir essa intrusão. Neste exemplo a intrusão foi detectada de forma reativa, como pode ser observado na linha 7:

```
1. |?-start_ HyLAA_agent.
2. * HyLAA Hybrid Agent waiting new perceptions…*
3. # Abolishing f:\HyLAA_Agent\ ontoid.pl
4. # 0.969 seconds to consult f:\HyLAA_Agent\ontoid.pl
5. New Case Problem: MonitoredPackage_3
6. Intrusion type: Back
7. Recommended solution (reactive): *Configure individual hosts and routers to not respond to ICMP requests or broadcast
```

ANEXO 1. Tutorial para instalação da biblioteca Thea

UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE
GRUPO DE ENGENHARIA DE SOFTWARE E ENGENHARIA DO CONHECIMENTO – GESEC

TUTORIAL PARA INSTALAÇÃO DA BIBLIOTECA **THEA – Versão Windows**

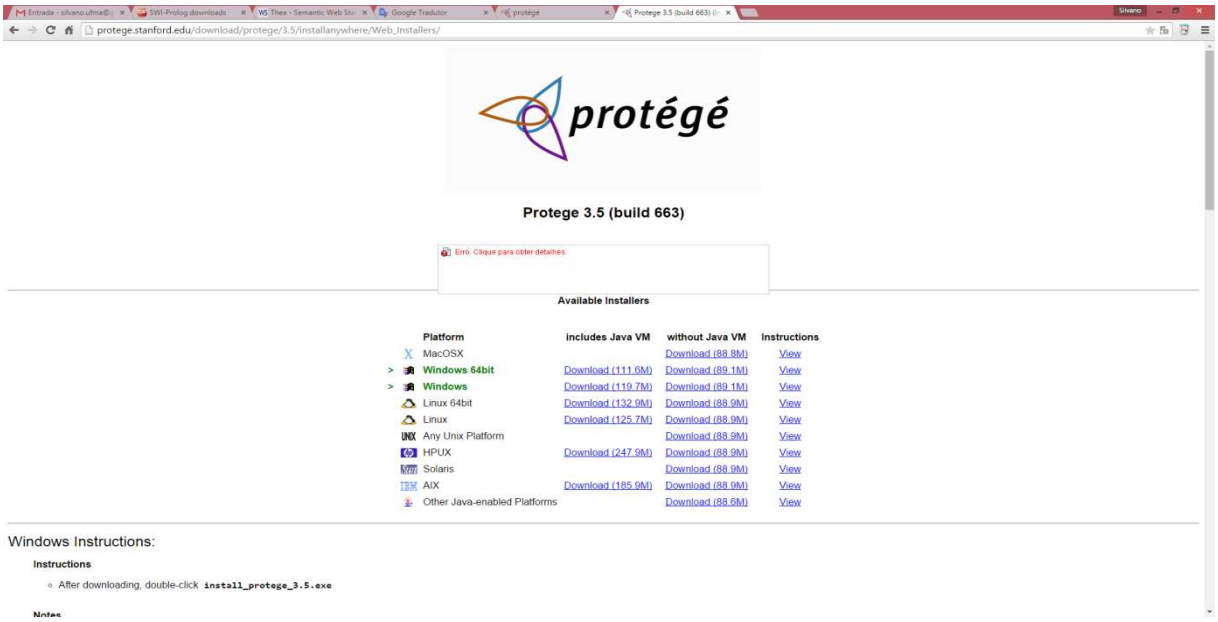
Autores: Silvano De Jesus Cantanhede de Oliveira (Versão Windows) e Adriana Leite Costa (Versão Linux)

Thea é uma biblioteca Prolog para gerar e manipular conteúdo OWL (Web Ontology Language), onde a mesma usa o analisador de Web Semântica do SWI-Prolog a fim de converter as ontologias do padrão OWL em fatos Prolog do tipo *.PL. E, para isso, o primeiro passo seria a criação das ontologias no padrão OWL através do software Protégé.

Usaremos a versão do Protégé 3.5 (escolher o padrão do Sistema Operacional, se 32bits ou 64bits), encontrado no link:

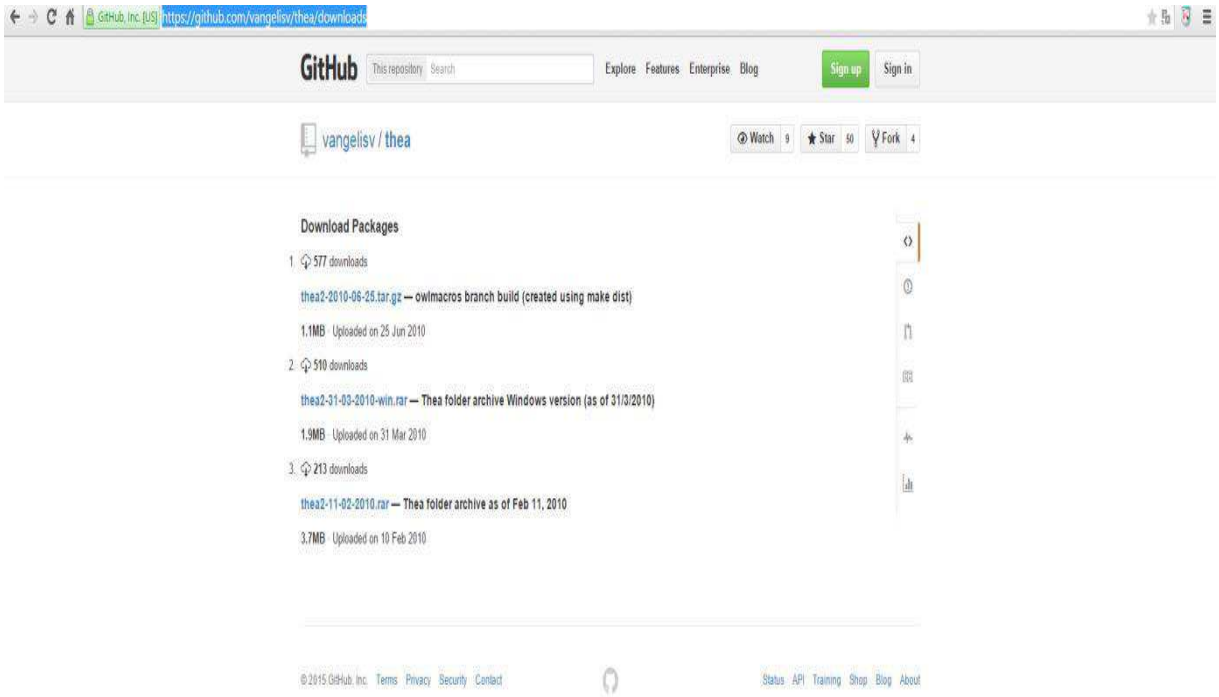
http://protege.stanford.edu/download/protege/3.5/installanywhere/Web_Installers/

E como o Protégé foi criado como uma aplicação Java, a máquina JVM (Java Virtual Machine) deverá estar instalada.



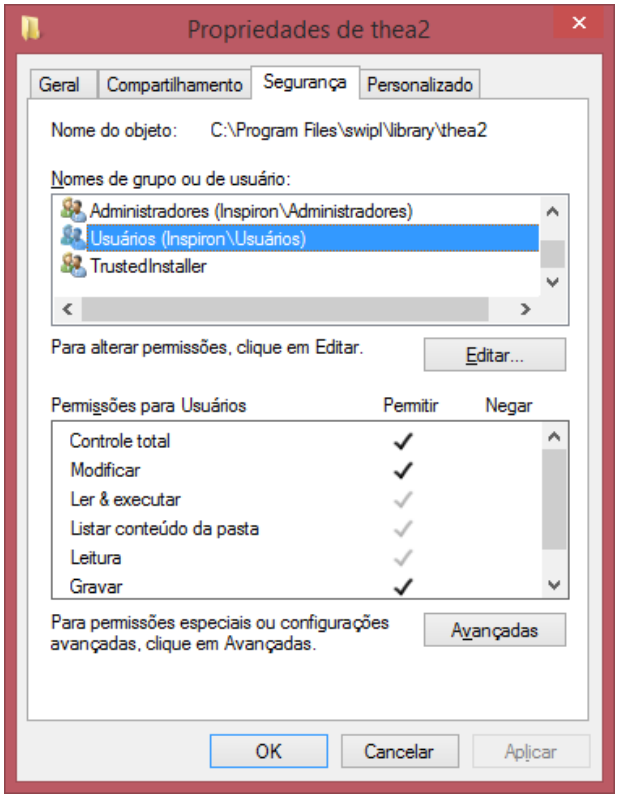
A biblioteca THEA não tem instalador, e sim um arquivo compactado com toda a estrutura de arquivos pronta para a implantação da mesma, onde deverá ser baixada no link: https://github.com/vangelisv/thea/downloads

E aqui será escolhida a versão de acordo com seu Sistema Operacional.

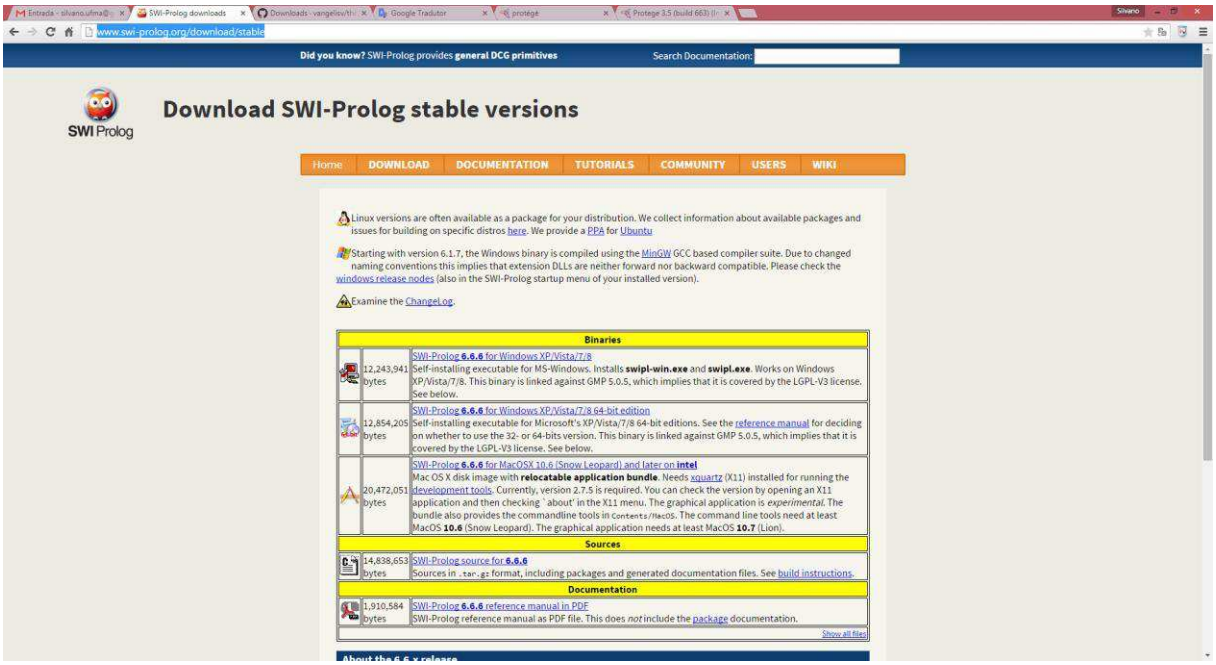


Após baixar o arquivo compactado, o mesmo deverá ser descompactado com a estrutura intacta em uma pasta de fácil acesso, ou mesmo dentro da pasta de instalação do SWI-Prolog. Por exemplo:
C:\Program Files\swipl\library

Onde, no Windows, a pasta thea2 deverá ter suas propriedades de segurança mudadas para Controle total do Acesso em usuários do sistema. Conforme figura a seguir:



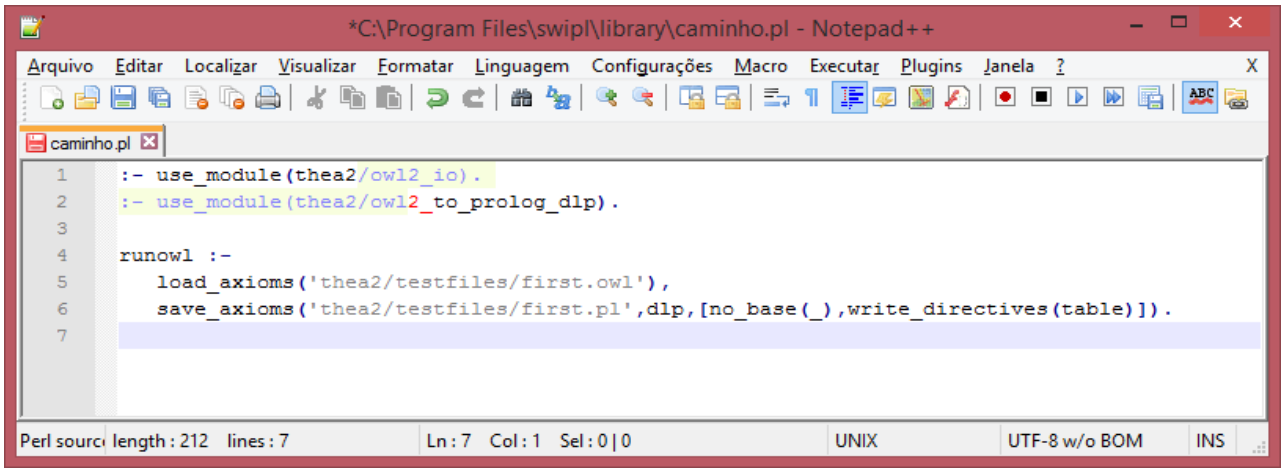
E mesmo sem precisar de um instalador para a biblioteca THEA, o software SWI-Prolog é necessário para executar o THEA. Portanto, segue o link de download do SWI-Prolog:
http://www.swi-prolog.org/download/stable



Onde na tela acima deverá ser escolhida a versão a ser baixada de acordo com o Sistema Operacional usado.

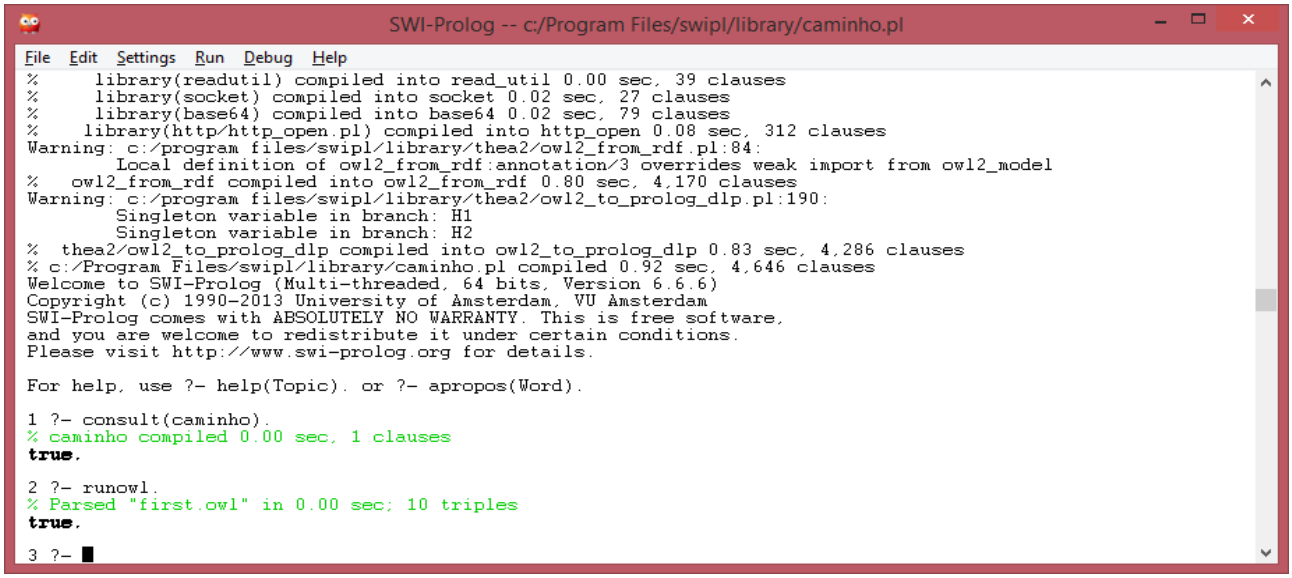
Para executar a aplicação e converter uma ontologia OWL para PL, os seguintes passos deverão ser feitos:

1. Criar um arquivo de nome caminho.pl, na pasta C:\Program Files\swipl\library, com o seguinte conteúdo:

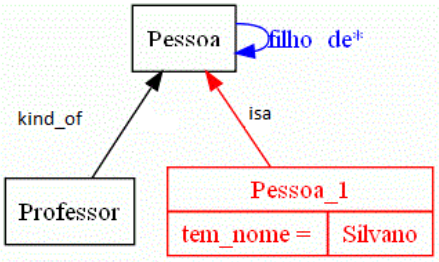


2. Trocar o nome dos arquivos .owl e .pl, das linhas 5 e 6, para os que estiver usando.
3. Colocar o arquivo *.owl na pasta thea2\testfiles.
4. Executar o arquivo caminho.pl a partir do iniciador de arquivos como SWI-Prolog.
5. Em seguida, digitar os seguintes comandos de dentro do SWI-Prolog:
6. consult(caminho).

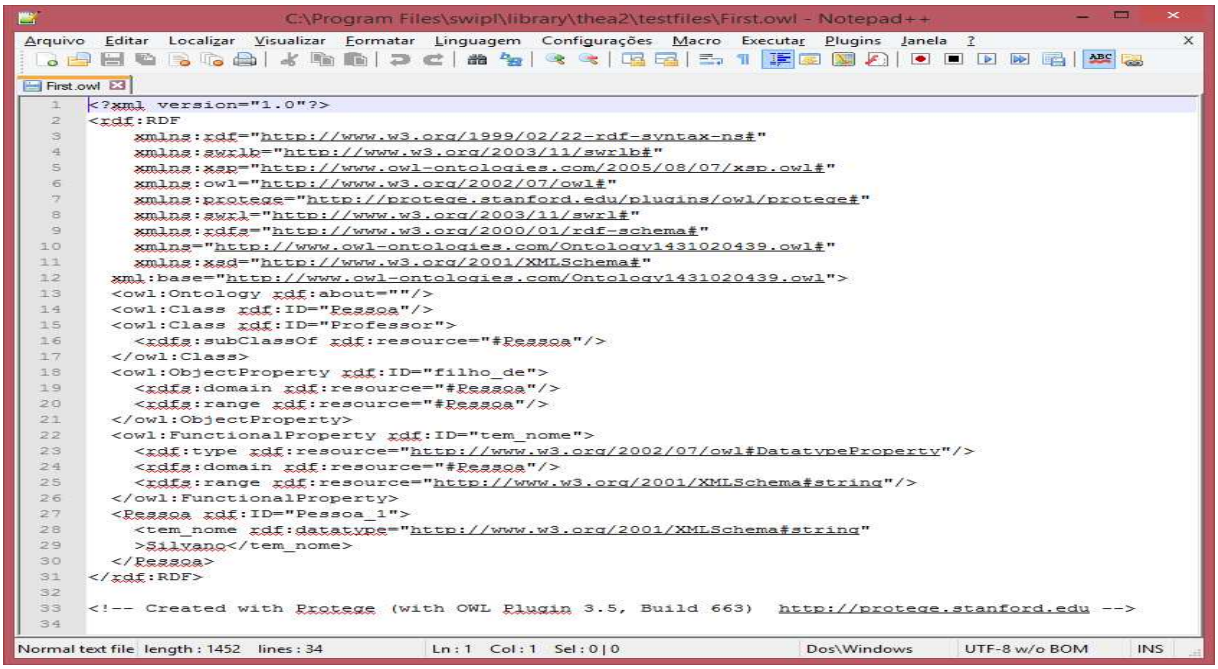
7. runowl.
8. Verificar se o resultado será True, e se o arquivo .pl foi criado na pasta testfiles.



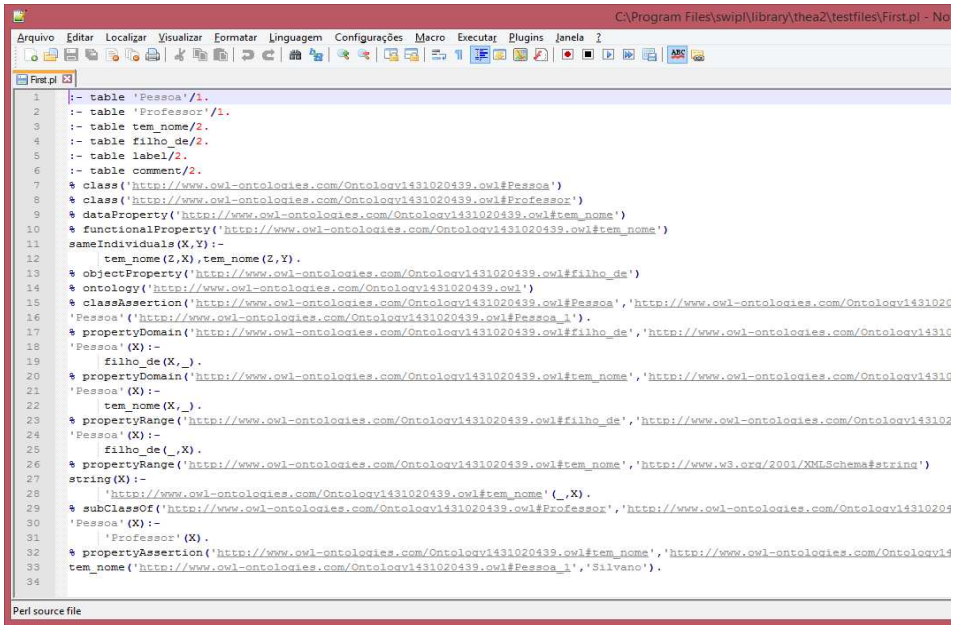
Por exemplo, na ontologia abaixo representada em frames:



A ontologia OWL gerada pelo Protégé é a seguinte:

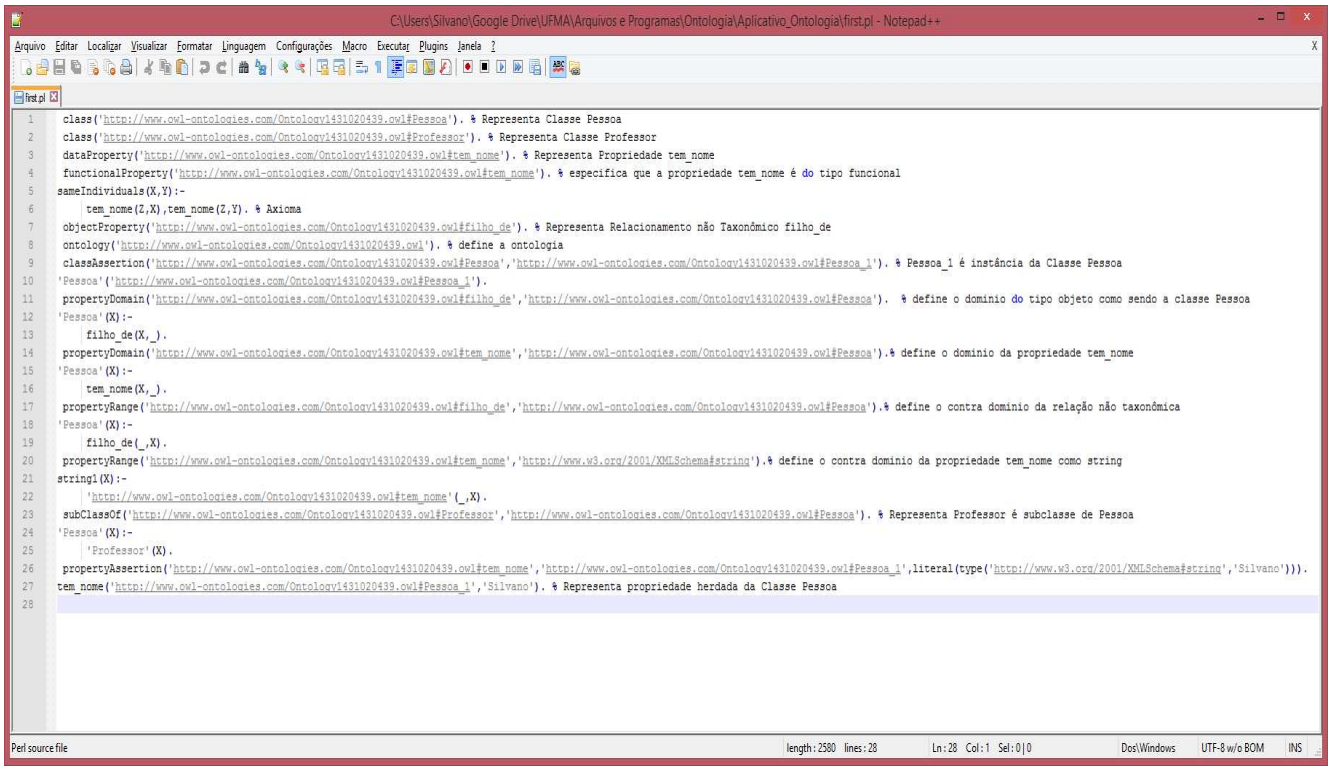


Após aplicarmos o THEA, o resultado em .pl é o seguinte:



Porém, para que o Win-Prolog possa entender o resultado, será necessário fazer uma limpeza de alguns caracteres a fim de que o mesmo possa ser executado.

Segue abaixo o resultado de acordo com o Win-Prolog:



Segue abaixo a transcrição da figura acima:

class('http://www.owl-ontologies.com/Ontology1431020439.owl#Pessoa'). **% Representa Classe Pessoa**

class('http://www.owl-ontologies.com/Ontology1431020439.owl#Professor'). **% Representa Classe Professor**

dataProperty('http://www.owl-ontologies.com/Ontology1431020439.owl#tem_nome'). **% Representa Propriedade tem_nome**

functionalProperty('http://www.owl-ontologies.com/Ontology1431020439.owl#tem_nome'). **% especifica que a propriedade tem_nome é do tipo funcional**

sameIndividuals(X,Y):-

tem_nome(Z,X),tem_nome(Z,Y). **% Axioma**

objectProperty('http://www.owl-ontologies.com/Ontology1431020439.owl#filho_de'). **% Representa Relacionamento não Taxonômico filho_de**

ontology('http://www.owl-ontologies.com/Ontology1431020439.owl'). **% define a ontologia**

classAssertion('http://www.owl-ontologies.com/Ontology1431020439.owl#Pessoa','http://www.owl-ontologies.com/Ontology1431020439.owl#Pessoa_1'). **% Pessoa_1 é instância da Classe Pessoa**

'Pessoa'('http://www.owl-ontologies.com/Ontology1431020439.owl#Pessoa_1').

propertyDomain('http://www.owl-ontologies.com/Ontology1431020439.owl#filho_de','http://www.owl-ontologies.com/Ontology1431020439.owl#Pessoa'). **% define o dominio do tipo objeto como sendo a classe Pessoa**

'Pessoa'(X):-e

filho_de(X,_).

propertyDomain('http://www.owl-ontologies.com/Ontology1431020439.owl#tem_nome','http://www.owl-ontologies.com/Ontology1431020439.owl#Pessoa'). **% define o dominio da propriedade tem_nome**

'Pessoa'(X):-

tem_nome(X,_).

propertyRange('http://www.owl-ontologies.com/Ontology1431020439.owl#filho_de','http://www.owl-ontologies.com/Ontology1431020439.owl#Pessoa'). **% define o contra domínio da relação não taxonômica**

'Pessoa'(X):-

filho_de(_,X).

propertyRange('http://www.owl-ontologies.com/Ontology1431020439.owl#tem_nome','http://www.w3.org/2001/XMLSchema#string'). **% define o contra domínio da propriedade tem_nome como string**

string1(X):-

'http://www.owl-ontologies.com/Ontology1431020439.owl#tem_nome'(_,X).

subClassOf('http://www.owl-ontologies.com/Ontology1431020439.owl#Professor','http://www.owl-ontologies.com/Ontology1431020439.owl#Pessoa'). **% Representa Professor é subclasse de Pessoa**

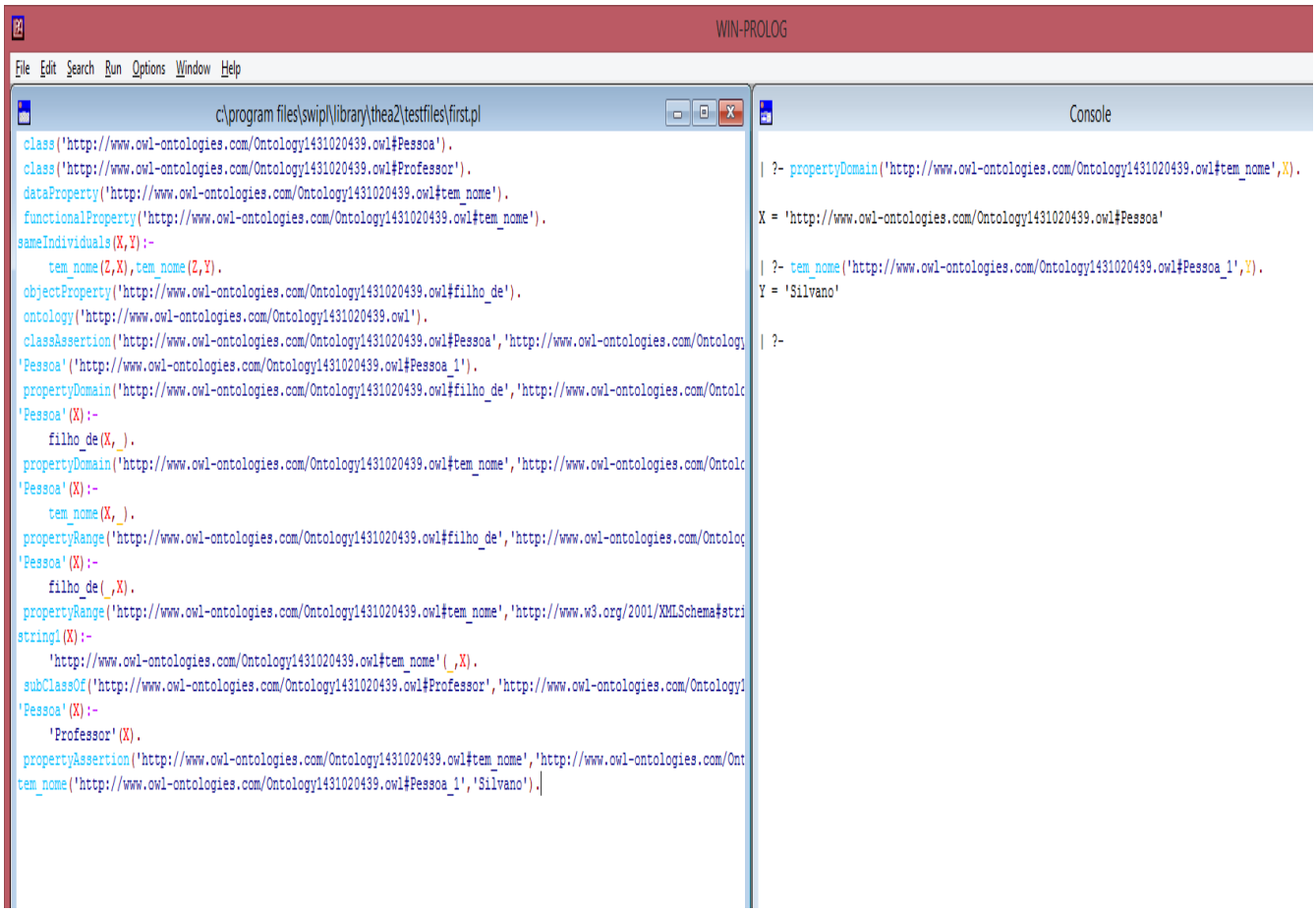
'Pessoa'(X):-

'Professor'(X).

propertyAssertion('http://www.owl-ontologies.com/Ontology1431020439.owl#tem_nome','http://www.owl-ontologies.com/Ontology1431020439.owl#Pessoa_1',literal(type('http://www.w3.org/2001/XMLSchema#string','Silvano'))).

tem_nome('http://www.owl-ontologies.com/Ontology1431020439.owl#Pessoa_1','Silvano'). **% Representa propriedade herdada da Classe Pessoa**

Em Win-Prolog aplicamos uma consulta para verificação da consistência do arquivo .pl gerado, conforme ilustrado na figura a seguir.



INSTALAÇÃO DA BIBLIOTECA **THEA – Versão Linux**

Passos para converter um arquivo OWL para Prolog usando o thea no Linux:

1. Instalar uma versão do Prolog. Nesse exemplo, instalamos o XPCE.
2. Colocar o arquivo com extensão OWL em um diretório do sistema operacional ("nomeArquivo.owl"), por exemplo: /home/adriana/thea2/testfiles
3. Editar o arquivo caminho.pl que contém o script de execução do Thea para o nome do arquivo OWL que está sendo mapeado (parte destacada na imagem abaixo).

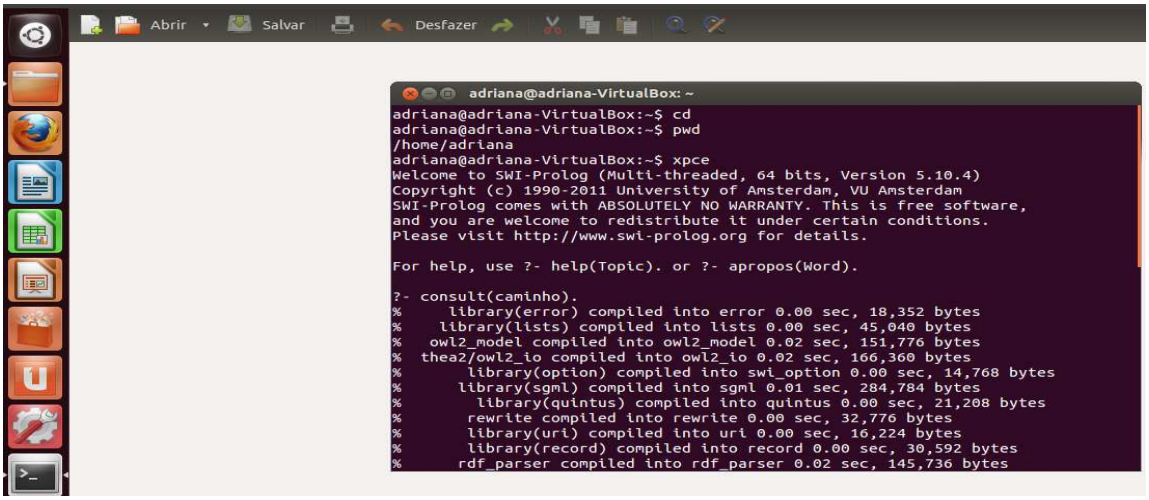
Entrar no Terminal do Ubuntu e digitar os seguintes comandos (conforme demonstrado nas telas seguintes):

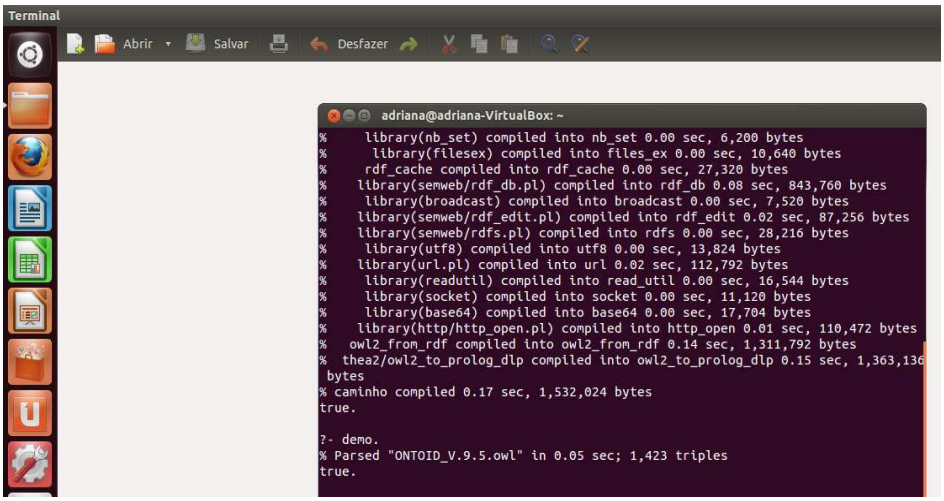
cd /home/adriana

xpce

consult(caminho).

demo.





O arquivo owl (" nomeArquivo.owl") vai ser gerado dentro do diretório: /home/adriana/