UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

FRANCISCO DANIEL LIMA DA SILVA

IMPLEMENTANDO ADAPTAÇÃO EM AMBIENTES INTELIGENTES UTILIZANDO SISTEMAS MULTIAGENTES E APRENDIZADO POR REFORÇO

QUIXADÁ

2019

FRANCISCO DANIEL LIMA DA SILVA

IMPLEMENTANDO ADAPTAÇÃO EM AMBIENTES INTELIGENTES UTILIZANDO SISTEMAS MULTIAGENTES E APRENDIZADO POR REFORÇO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Marcos Antonio de Oliveira

QUIXADÁ

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S58i Silva, Francisco Daniel Lima da.
Implementando adaptação em ambientes inteligentes utilizando sistemas multiagentes e aprendizado por reforço / Francisco Daniel Lima da Silva. – 2019.
63 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2019.
Orientação: Prof. Dr. Marcos Antonio de Oliveira.

1. Multiagent Systems. 2. Aprendizado do computador. 3. Algoritmos. 4. Ambientes Inteligentes. I. Título.

CDD 005

FRANCISCO DANIEL LIMA DA SILVA

IMPLEMENTANDO ADAPTAÇÃO EM AMBIENTES INTELIGENTES UTILIZANDO SISTEMAS MULTIAGENTES E APRENDIZADO POR REFORÇO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Aprovada em: ____/____/_____

BANCA EXAMINADORA

Prof. Dr. Marcos Antonio de Oliveira (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dra. Maria Viviane de Menezes
Universidade Federal do Ceará (UFC)

Prof. Dr. Regis Pires Magalhães
Universidade Federal do Ceará (UFC)

A Deus.

À minha Mãe, Maria de Fátima, pilar da minha formação como ser humano. Mãe, devo a você tudo o que sou.

AGRADECIMENTOS

Agradeço à minha mãe, Maria de Fátima de Lima Silva, pelo apoio durante esses quatro anos de graduação em que nunca deixou me faltar nada e sempre me incentivou.

À minha família, em especial, minhas irmãs Larissa Lima e Irisleide Lima, que estiveram comigo durante essa caminha.

Aos amigos que fiz durante minha graduação, em especial Jessica Nunes, Sabrina Cunha, Fernanda Albuquerque e Jairo Márcio. Obrigado pelo carinho e apoio.

Agradeço ao Prof. Dr. Marcos Antonio de Oliveira, por me orientar neste trabalho e pela sua dedicação e paciência ao longe deste ano. Aos professores participantes da banca examinadora, Prof. Dra. Maria Viviane de Menezes e Prof. Dr. Regis Pires Magalhães, pelas contribuições que foram imprescindíveis para a construção deste trabalho.

Aos colegas da turma, pelas reflexões, críticas, sugestões recebidas e conhecimento compartilhado durante essa jornada.

E a todos que direta e indiretamente fizeram parte da minha graduação.

“Who knows what miracles
You can achieve
When you believe somehow you will
You will when you believe.”

(Billy Strange e Mac Davis)

RESUMO

Com o avanço da tecnologia, os serviços de controle de ambientes têm tornado-se cada vez mais inteligentes e autônomos. Para construir sistemas inteligentes é preciso considerar situações de mudanças, onde o sistema precisa adaptar-se aos novos comportamentos observados no ambiente. O presente trabalho tem como objetivo implementar um sistema para controle de casas inteligentes utilizando o paradigma de Sistemas Multiagentes e técnicas de Aprendizado por Reforço, de maneira que o sistema possa adaptar-se as preferências dos usuários e possa reagir às mudanças dos comportamento desses usuários.

Neste contexto, este trabalho propõe o uso de uma técnica de Aprendizagem por Reforço - especificamente o Algoritmo *Q-learning* - como uma estratégia de aprendizado e utilização de uma ferramenta para o desenvolvimento de Sistemas Multiagentes. Para tanto, foram criadas a arquitetura proposta para a casa inteligente e suas funcionalidades. Uma análise foi realizada para a seleção das ferramentas de desenvolvimento e a implementação do sistema foi efetuada. Além disto, neste trabalho é apresentado um estudo de caso para o cenário de adaptação a um habitante de uma residência, objetivando avaliar e demonstrar o uso das funcionalidades desenvolvidas.

Palavras-chave: Sistemas Multiagentes. Aprendizado por Reforço. Algoritmo Q-learning. Ambientes Inteligentes.

ABSTRACT

With the advancement of technology, environment control services have become increasingly intelligent and autonomous. To build intelligent systems we need to take into consideration changing situations where the system needs to adapt to the new behaviors observed in the environment. The present work aims to implement a smart home control system using the paradigm of Multiagent Systems and Reinforcement Learning techniques, so that the system can adapt to user preferences and react to changes in user behavior.

In this context, this work proposes the use of a Reinforcement Learning technique - specifically the Q-learning Algorithm - as a learning strategy and use of a framework for the development of Multiagent Systems. Therefore, the proposed architecture for the smart home and its functionalities were created. An analysis was performed for the selection of development framework and the system implementation was performed. In addition, this paper presents a case study for the adaptation scenario to an inhabitant of a residence, aiming to evaluate and demonstrate the use of the developed features.

Keywords: Multi-agent Systems. Reinforcement Learning. Q-learning Algorithm. Ambient Intelligence.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AM	Aprendizado de Máquina (*Machine Learning*)
AmI	Ambientes Inteligentes (*Ambient Intelligence*)
BURLAP	*Brown-UMBC Reinforcement Learning and Planning*
CArtAgO	*Common ARTifact infrastructure for AGents Open environments*
FIPA	*Foundation for Intelligent Physical Agents*
GUI	*Graphical User Interface*
IoT	Internet das Coisas (*Internet of Things*)
JADE	*Java Agent Development Framework*
JSON	*JavaScript Object Notation*
PQLA	*Planning Q-learning Algorithm*
RL	Aprendizado por Reforço (*Reinforcement Learning*)
SMA	Sistemas Multiagentes

SUMÁRIO

1 INTRODUÇÃO

Cada vez mais almeja-se sistemas que possam antecipar nossas necessidades e cooperar conosco. A área da Inteligência Artificial tem estudado e desenvolvido técnicas e ferramentas com capacidade de processamento simbólico a fim de encontrar métodos genéricos para automatizar atividades perceptivas, cognitivas e manipulativas (PEREIRA, 1988).

Atualmente, graças a redução dos custos e o aumento da oferta de objetos com capacidade de sensoriamento, processamento e comunicação, tornou-se possível oferecer componentes economicamente viáveis para aplicação de conceitos da inteligência artificial em atividades do cotidiano (SANTAELLA *et al.*, 2013).

Nos últimos anos, a Internet das Coisas (*Internet of Things*) (IoT) tem recebido bastante atenção devido ao seu potencial de uso nas mais diversas áreas das atividades humanas, tais como cidades inteligentes, saúde e automação de ambientes (SANTOS *et al.*, 2016). O termo é utilizado para referir-se a ideia de uma rede mundial de objetos conectados que coletam dados e podem comunicar-se entre si.

De acordo com Santos *et al.* (2016), a conexão com a rede mundial de computadores viabilizará controlar remotamente objetos eletrônicos e permitirá que os próprios objetos sejam acessados como provedores de serviços, gerando um grande número de oportunidades tanto no âmbito acadêmico quanto no industrial.

Nesse contexto, surgem os Ambientes Inteligentes (*Ambient Intelligence*) (AmI) (AARTS; WICHERT, 2009) — que são ambientes eletrônicos sensíveis e adaptáveis, que respondem às ações de pessoas e objetos e atendem às suas necessidades. Ambientes Inteligentes utilizam técnicas computacionais de várias áreas da computação, como a computação ubíqua ou pervasiva e pesquisas de sistemas inteligentes que fornecem algoritmos de aprendizagem, adaptadores de padrões, classificação de gestos e avaliação de situação (SHADBOLT, 2003).

Segundo Ramos *et al.* (2008), esses ambientes devem estar cientes das necessidades dos seus habitantes, personalizando suas necessidades e prevendo comportamentos. Os ambientes AmI podem ser de diversos tipos, como residências, escritórios, salas de reunião, escolas, hospitais, centros de controle, veículos, lojas e aparelhos digitais (RAMOS *et al.*, 2008).

Em Zambiasi *et al.* (2002), é demonstrada a importância e a necessidade de ambientes inteligentes para diferentes grupos de pessoas. Os autores exemplificam a utilização de automação de ambientes para propiciar comodidade e conforto às pessoas inseridas nestes ambientes e cita os casos em que pessoas com necessidades especiais ou com deficiências físicas podem

necessitar de ambientes automatizados, de forma a facilitar sua interação com o ambiente e uma maior independência pessoal.

Em ambientes dinâmicos, onde há mudanças constantes nos cenários e desejos das pessoas inseridas nesse ambiente, nem sempre é aplicável a utilização de cenários de automatização preestabelecidos. Em casos como esses, o ambiente nem sempre irá agir conforme se deseja, pois ele pode não conseguir se adaptar aos novos desejos ou não se adaptará a uma certa pessoa ou a um determinado grupo de pessoas que adentrem o ambiente.

1.1 Objetivos

1.1.1 Objetivo Geral

Definir e implementar um modelo de AmI baseado no paradigma de Sistemas Multiagentes (SMA), utilizando a técnica de Aprendizado de Máquina, conhecida por Aprendizado por Reforço (*Reinforcement Learning*) (RL), para alterar planos de agentes em tempo de execução baseado em experiências passadas, buscando assim a auto-adaptação do sistema.

1.1.2 Objetivos Específicos

Considerando o desenvolvimento do trabalho e o objetivo geral apresentado, destacam-se os seguintes objetivos específicos:

- Modelar a aplicação e definir cenários e componentes a serem desenvolvidos;
- Analisar e selecionar um algoritmo de RL que possa ser aplicado em conjunto com um SMA para gerar e/ou alterar planos de ação em tempo de execução;
- Analisar e selecionar uma plataforma para desenvolvimento de Sistemas Multiagentes;
- Implementar um ambiente simulado de casa automatizada inteligente utilizando a plataforma e algoritmo selecionados.
- Analisar os resultados obtidos.

1.2 Organização

O presente trabalho está organizado da seguinte forma: no Capítulo 2 é apresentado o referencial teórico englobando alguns conceitos sobre Ambientes Inteligentes, Agentes, Sistemas Multiagentes e Aprendizado por Reforço. Já no Capítulo 3 são apresentados os trabalhos

relacionados e as suas semelhanças e diferenças com o presente trabalho. No Capítulo 4 são apresentados os passos da execução deste trabalho e os resultados obtidos. Por fim, o Capítulo 5 apresenta as considerações finais deste trabalho, propondo a realização de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A fim de garantir a base de informação que sustentará a discussão proposta neste trabalho, é necessário conceituar alguns tópicos fundamentais relacionados ao tema para proporcionar um maior entendimento sobre o assunto. A Seção 2.1 apresenta a definição de Ambientes Inteligentes, a Seção 2.2 apresenta a definição, propriedades e tipos de agentes, na Seção 2.4 são apresentados conceitos sobre Sistemas Multiagentes, na Seção 2.5 são apresentados conceitos sobre Aprendizado de Máquina e algumas de suas técnicas.

2.1 Ambientes Inteligentes

Ambientes Inteligentes - ou *Ambient Intelligence (AmI)* - são ambientes em que seus habitantes são auxiliados por sistemas computacionais que identificam sua presença e atendem suas necessidades de uma maneira que tais habitantes não percebem que estão interagindo com um sistema computacional.

AmI incorpora aspectos da computação ubíqua e computação pervasiva, além de pesquisa da área da inteligência artificial, abrangendo contribuições de aprendizado de máquina, sistemas multiagentes e robótica (COOK *et al.*, 2009).

Segundo Aarts e Wichert (2009), AmI são ambientes eletrônicos sensíveis e adaptáveis, que respondem às ações de pessoas e objetos e atendem às suas necessidades, proporcionando uma maior eficiência, conforto e comodidade para seus habitantes.

Acrescentando essa ideia, Gaggioli (2005 apud GUERRA, 2007) define um ambiente inteligente como:

> Um ambiente digital sensível e que responde à presença de pessoas. A visão de um ambiente inteligente se baseia em duas características: inteligência e inclusão. Inteligência pelo fato de o ambiente poder analisar o contexto e se adaptar às pessoas e objetos que se encontram no ambiente, aparentando inclusive emoções. A inclusão se refere a que pequenos dispositivos computacionais sejam parte das atividades diárias dos usuários sem que sejam percebidos. (GAGGIOLI, 2005 apud GUERRA, 2007)

Para Friedewald *et al.* (2005), um ambiente inteligente deve reconhecer seus habitantes e algumas de suas necessidades e desejos, bem como mudanças no comportamento, necessidades e desejos de tais habitantes. O ambiente deve responder às ações e presença dos indivíduos de maneira harmoniosa, discreta e às vezes de maneira invisível, ainda assim permanecendo sob o controle dos humanos.

Para desenvolvimento deste trabalho, tomam-se como base os conceitos de Ambientes Inteligentes de Gaggioli (2005) e Friedewald *et al.* (2005), segundo os quais o ambiente deve analisar o contexto e adaptar-se às pessoas, reconhecendo seus habitantes e algumas de suas necessidades e desejos e mudanças de comportamento.

2.2 Agentes

Antes de abordar sistemas multiagentes, é necessário revisar os principais conceitos de agente.

2.2.1 Definição de Agente

Existem inúmeras definições de agente, e embora não exista uma definição universalmente aceita do termo agente, existe um consenso geral de que a autonomia é essencial num agente.

Para Weiss (1999), parte da dificuldade de definir o termo agente é que os vários atributos associados a agentes são de importância diferente para domínios diferentes. Assim, para algumas aplicações, a capacidade dos agentes de aprender com suas experiências é de suma importância; para outras aplicações, a aprendizagem não é apenas sem importância, é indesejável. (WEISS, 1999).

Uma definição de agentes, proposta por Russell e Norvig (2016), define que um agente é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores.

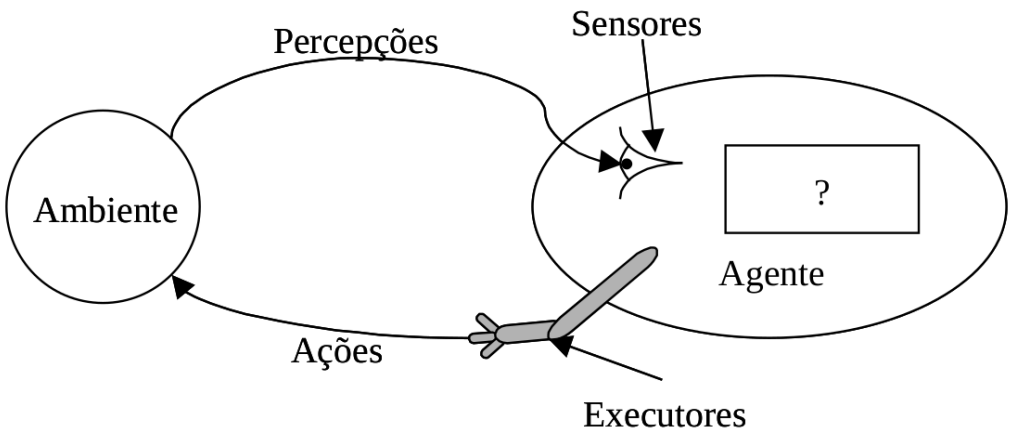
O trabalho de Reis (2003) caracterizou um agente como um sistema computacional que possui capacidade de percepção do ambiente que está situado através do uso de sensores e capacidade de decisão, podendo agir nesses ambientes de forma autônoma utilizando seus atuadores. Para Reis (2003), um agente também deve possuir capacidade de comunicação com outros agentes e/ou humanos para executar a função para a qual foi projetado.

Ainda segundo Reis (2003), independentemente do tipo de agente e ambiente, é essencial a capacidade do agente de tomar conhecimento do ambiente e nele agir de forma autônoma. Sendo assim, o agente deve possuir sensores e atuadores apropriados ao seu ambiente e à execução das tarefas para as quais foi projetado.

Na Figura 1 temos uma visão esquemática do funcionamento de um agente. O

agente, representado por um robô, usa suas mãos como executores de ações no ambiente e seus sensores para capturar percepções desse mesmo ambiente.

Figura 1 - Exemplo de interação de um agente com ambiente



Fonte: Reis (2003)

Um exemplo que Russell e Norvig (2016) apontam, pode mostrar de maneira mais clara a definição de sensores e atuadores em diferentes tipos de agentes. Por exemplo, um agente humano tem olhos, ouvidos e outros órgãos como sensores, e tem mãos, pernas, boca e outras partes do corpo que servem como atuadores. Um agente robótico pode ter câmeras e detectores da faixa de infravermelho funcionando como sensores e vários motores como atuadores. Um agente de software recebe sequências de teclas digitadas, conteúdo de arquivos e pacotes de rede como entradas sensoriais e atua sobre o ambiente exibindo algo na tela, escrevendo em arquivos e enviando pacotes de rede (RUSSELL; NORVIG, 2016).

Analisando as definições aqui descritas, observa-se que o conceito de agente é muito amplo e que não existe um consenso acerca do seu significado. Entretanto, de acordo com estas definições, pode-se notar alguns atributos básicos para o desenvolvimento de agentes.

Para seguimento do desenvolvimento deste trabalho, tomam-se como base os principais atributos associados a agentes que são importantes para o presente trabalho, segundo os quais um agente pode ser caracterizado como uma entidade real ou virtual, que possui comportamentos autônomos e que está inserido em um ambiente, podendo perceber (sensores), agir (atuadores), deliberar e comunicar-se com outros agentes.

2.2.2 Características dos Agentes

Um agente pode ser definido a partir de um conjunto de características básicas. Uma das definições de agente mais conhecidas e aceitas na comunidade é apresentada por Wooldridge e Jennings (1995), que consideram que agentes inteligentes podem ser caracterizados pela presença das seguintes propriedades:

- **Autonomia:** Os agentes possuem total controle sobre si e sobre suas ações, não sofrendo intervenção direta de humanos ou outros agentes;
- **Capacidade de Reação:** Os agentes mantém uma interação contínua com o ambiente em que está inserido e, ao perceber mudanças no seu ambiente, são capazes de responder às alterações que nele ocorrem em um tempo adequado para satisfazer seus objetivos.
- **Capacidade Proativa:** Os agentes não se limitam a agir em resposta a alterações do seu ambiente. Eles são capazes de tomar decisões e exibir comportamento direcionado a objetivos ao tomar a iniciativa de tentar satisfazê-los;
- **Habilidade Social:** Os agentes podem interagir com outros agentes (e eventualmente com humanos), sendo necessária a habilidade de comunicação e uma linguagem de comunicação comum aos agentes. Foi a partir daí que surgiram os sistemas multiagentes.

De acordo com Reis (2003), além das propriedades apresentadas, também são referidas como propriedades comuns dos agentes:

- **Inteligência:** O estado de um agente é formalizado por conhecimento (i.e. crenças, objetivos, planos e aceitações) e ele interage com outros agentes utilizando uma linguagem simbólica. Possui capacidade de raciocínio abstrato e de resolução de novos problemas e adaptação a novas situações;
- **Capacidade de Aprendizagem:** Capacidades de aprendizagem que fazem com que o agente adquira novo conhecimento e altere o seu comportamento baseado na sua experiência prévia.

A escolha de quais as características que devem estar presentes em um dado agente depende da funcionalidade de cada agente.

Segundo Andrade *et al.* (2016), no contexto de casas inteligentes, observa-se que os conceitos de Autonomia e Capacidade de Reação são os mais importantes, apesar dos outros conceitos terem sua fundamental importância. A utilização de agentes com essas características permite a criação de um sistema que não precisa ser manipulado por terceiros, e que suas ações sejam baseadas em mudanças no ambiente delimitado por uma casa inteligente (ANDRADE *et*

al., 2016).

No contexto deste trabalho, pretende-se que cada agente seja capaz de realizar a sua tarefa de forma autônoma, inteligente e com capacidade de aprendizagem. No sistema aqui proposto, os agentes desenvolvidos têm a capacidade de perceber informações sobre o ambiente em que está inserido e de agir sobre esse ambiente. A propriedade de Capacidade de Aprendizagem também é explorada com o desenvolvimento de agentes que possuem capacidade de aprender e alterar seu comportamento com base nas ações já realizadas pelos habitantes do ambiente em que o agente está inserido.

2.3 Agentes Cognitivos ou Deliberativos

Os agentes cognitivos são baseados nos modelos de organizações sociais, em termos de sociedades humanas como grupos, hierarquias, mercados, etc. Segundo Sichman (2003), esses agentes cognitivos possuem uma representação explícita do ambiente e dos outros agentes e podem raciocinar sobre as ações tomadas no passado e planejar as futuras ações.

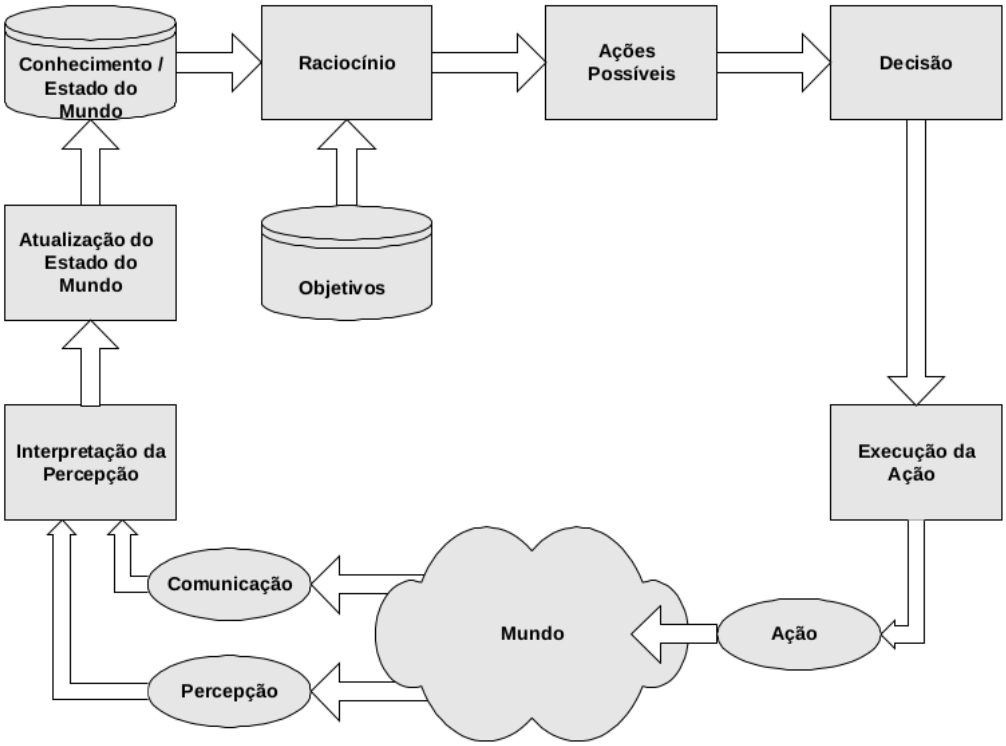
De acordo com Sichman (2003), os agentes cognitivos podem ainda interagir com os demais membros da comunidade através de linguagens e protocolos de comunicação e estratégias de negociação.

As arquiteturas Deliberativas seguem a abordagem clássica da Inteligência Artificial, sendo fortemente baseadas no paradigma simbólico. Tais arquiteturas são bastante utilizadas para o desenvolvimento de agentes cognitivos (SICHMAN, 2003). Segundo Reis (2003), essas arquiteturas interpretam os agentes como parte de um sistema baseado em conhecimento.

Na Figura 2, Reis (2003) apresenta um esquema típico de uma arquitetura deliberativa, onde, após a interpretação da percepção proveniente do ambiente, o agente utiliza esta informação para manter atualizada uma representação interna, usualmente simbólica, do estado do mundo. Na arquitetura apresentada por Reis (2003), o estado do ambiente em conjunto com os objetivos do agente são utilizados como forma de selecionar as ações mais apropriadas para serem executadas.

Dentre as arquiteturas deliberativas mais utilizadas, destaca-se a denominada Arquitetura BDI (*"Belief, Desire, Intention"*), sugerida no *Stanford Research Institute*, na década de 80 (SICHMAN, 2003). A arquitetura BDI é amplamente explorada em agentes cognitivos e nos agentes desenvolvidos no presente trabalho. Os principais conceitos dessa arquitetura serão abordados na próxima seção.

Figura 2 - Esquema genérico de uma Arquitetura Deliberativa



Fonte: Reis (2003)

2.3.1 Arquitetura BDI

Existem diversas arquiteturas de agentes descritas na literatura. Uma arquitetura para o modelo deliberativo é a arquitetura BDI. As demais arquiteturas não são abordadas neste trabalho.

Segundo Reis (2003), a arquitetura BDI [Rao e Georgeff, 1991] é uma arquitetura essencialmente deliberativa, onde o estado interno de processamento de um agente é descrito através de um conjunto de "estados mentais". Assim, o estado do agente é representado por três estruturas: suas crenças (*beliefs*), seus desejos (*desires*) e suas intenções (*intentions*).

Em seu trabalho, Reis (2003) descreve cada um desses estados mentais:

- As crenças de um agente referem-se ao que o agente acredita em cada instante, e descrevem o estado do mundo do agente (o seu conhecimento sobre o ambiente). As crenças representam, desta forma, informação.
- Os desejos de um agente referem-se ao que o agente deseja obter. No entanto, a forma de alcançar esses desejos pode não ser conhecida em um dado instante. Os desejos podem ser, em um dado momento, inconsistentes. Os objetivos dos agentes resultam de um processo de raciocínio, por parte do agente, que consiste em uma escolha de um subconjunto dos

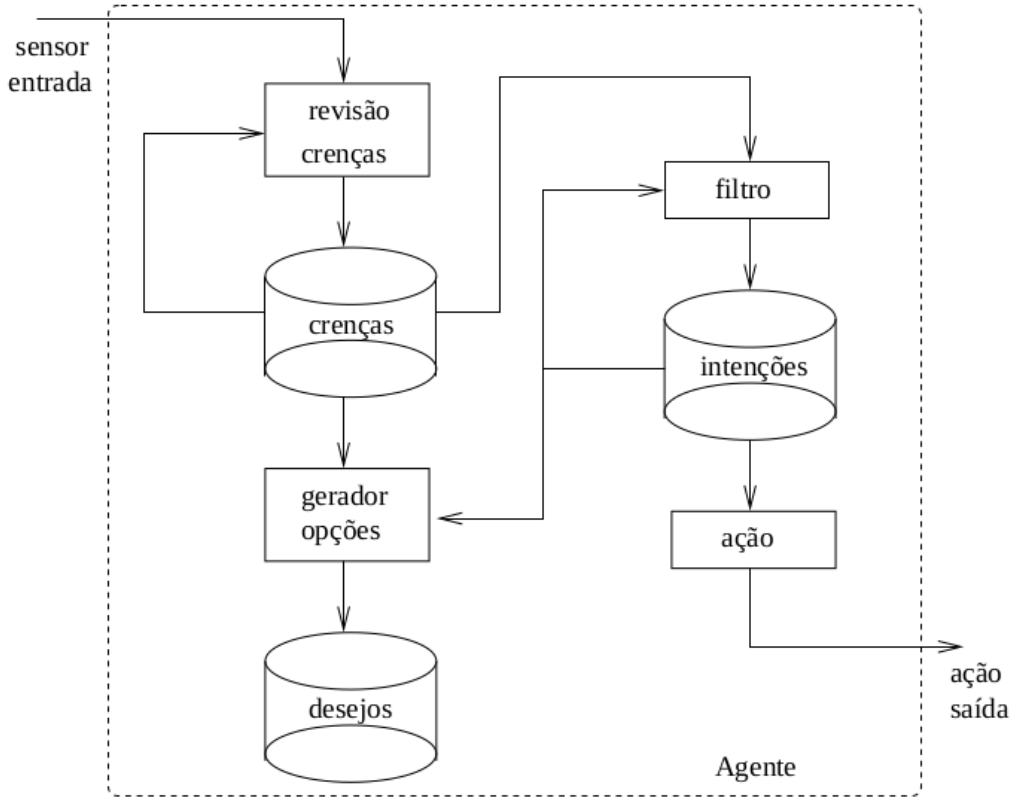
desejos que são consistentes e atingíveis.

- As intenções referem-se a um conjunto de ações ou tarefas que o agente selecionou, comprometendo-se assim na realização dos seus objetivos. As intenções devem ser consistentes internamente e representam o resultado do processo de deliberação.

De acordo com Girardi (2004), além destes componentes, algumas arquiteturas BDI usam o conceito de planos, que seriam o "passo a passo"a ser seguido, quando gerada uma intenção, para a realização de uma ação sobre o ambiente.

De forma esquemática, a arquitetura BDI genérica está apresentada na Figura 3, conforme proposto por Wooldridge (1999)

Figura 3 - Modelo Genérico da Arquitetura BDI



Fonte: Wooldridge (1999)

Em seu trabalho Hübner *et al.* (2004), descrevem em detalhes o funcionamaneto da arquitetura BDI. Como pode ser observado na Figura 3, a função de revisão de crenças percebe propriedades do ambiente através do uso de sensores e, consultando as crenças anteriores do agente, atualiza essas crenças para que elas reflitam o novo estado do ambiente. A função denominada na Figura 3 como Gerador Opções verifica quais as novas alternativas de estados a serem atingidos, que são relevantes para os interesses particulares daquele agente. Segundo

Hübner *et al.* (2004), isto deve ser feito com base no estado atual do mundo (conforme as crenças do agente) e nas intenções com que o agente já está comprometido. A atualização dos objetivos se dá, então, de duas formas: as observações do ambiente possivelmente determinam novos objetivos do agente, e a execução de intenções de mais alto nıvel pode gerar a necessidade de que objetivos mais especıficos sejam atingidos (HÜBNER *et al.*, 2004).

Em seguida, a função Filtro atualiza o conjunto de intenções do agente, com base nas crenças e desejos atualizados e nas intenções já existentes. De acordo com (HÜBNER *et al.*, 2004), esse processo realizado pela função Filtro para determinar como atualizar o conjunto de intenções do agente é normalmente chamado de deliberação. Com o conjunto de intenções já atualizado, a escolha de qual ação especıfica, entre aquelas pretendidas, será a próxima a ser realizada pelo agente no ambiente é feita pela função Ação (HÜBNER *et al.*, 2004).

2.4 Sistemas Multiagentes

Com o aumento do uso de sistemas complexos, há a necessidade de decompor o sistema em subsistemas de menor complexidade. A partir disso, uma solução é criada através do agrupamento de agentes que trabalham cooperativamente, cada um deles resolvendo parte do problema. A este agrupamento é dado o nome de Sistema Multiagente (SMA) (JUCHEM; BASTOS, 2001).

Os Sistemas Multiagente são sistemas compostos por múltiplos agentes, que exibem um comportamento autônomo, mas ao mesmo tempo interagem com os outros agentes presentes no sistema (REIS, 2003).

Em Wooldridge (2009), um sistema multiagente é classificado como um sistema que consiste de um número de agentes, que interagem entre si, geralmente trocando mensagens através de alguma infra-estrutura de rede de computadores. Ainda segundo Wooldridge (2009), para interagir com sucesso, esses agentes exigirão a capacidade de cooperar, coordenar e negociar uns com os outros, da mesma maneira que nós cooperamos, coordenamos e negociamos com outras pessoas em nossas vidas diárias.

De acordo com Girardi (2004), um SMA pode ser caracterizado como um grupo de agentes que atuam em conjunto no sentido de resolver problemas que estão além das suas habilidades individuais. Os agentes realizam interações entre eles de modo cooperativo para atingir uma meta.

Segundo Zambonelli, et. al. (2009), podemos distinguir os sistemas multiagentes em

duas principais classes:

- **Sistemas de Resolução Distribuída de Problemas**, nos quais os agentes envolvidos são explicitamente projetados para, de maneira cooperativa, atingirem um dado objetivo, considerando-se que todos eles são conhecidos à priori e supondo que todos são benevolentes, existindo desta forma confiança mútua em relação às suas interações; e
- **Sistemas Abertos**, nos quais os agentes não são necessariamente projetados para atingirem um objetivo comum, podendo ingressar e sair do sistema de maneira dinâmica. Nesse caso, a chegada dinâmica de agentes desconhecidos precisa ser levada em consideração, bem como a possível existência de comportamento não benevolente no curso das interações.

O trabalho aqui desenvolvido utilizou a abordagem definida por Andrade *et al.* (2016), que construiu um modelo de sistema multiagente para resolução distribuída de problemas, onde o objetivo do sistema é buscar uma melhor qualidade de vida para os habitantes de uma casa inteligente. Esta tarefa principal divide-se entre os agentes, que são responsáveis por manipular os componentes da casa.

2.4.1 Plataformas para Desenvolvimento de Sistemas Multiagentes

Existem diversas linguagens e plataformas para o desenvolvimento e implementação de Sistemas Multiagentes. Essas plataformas providenciam uma infraestrutura dentro da qual os agentes operam e interagem. Um dos principais aspectos dessas plataformas é o seu suporte para a comunicação entre agentes e o suporte a implementação de um ambiente em que os agentes estão situados. Dentre essas plataformas, as mais utilizadas para desenvolvimento de Sistemas multiagentes são:

Jason (JASON, 2007)[1]. A plataforma Jason fornece uma infra-estrutura de desenvolvimento de sistemas multiagentes baseada em um interpretador para uma versão estendida da linguagem *AgentSpeak(L)*. Ele implementa a semântica operacional da linguagem *AgentSpeak(L)* e fornece uma plataforma para o desenvolvimento de sistemas multiagente com muitos recursos personalizáveis pelo usuário.

JADE (JADE, 1999)[2]. O *Java Agent Development Framework* (JADE) é um

[1] http://jason.sourceforge.net/
[2] https://jade.tilab.com/

framework totalmente implementado na linguagem Java. Ele simplifica a implementação de sistemas multiagentes por meio de um *middleware* que atende às especificações do *Foundation for Intelligent Physical Agents* (FIPA) e fornece um conjunto de ferramentas gráficas que suportam as fases de depuração e implantação do sistema.

JACK (JACK, 2001)[3]. O JACK é um *framework* para o desenvolvimento de sistemas multiagentes baseado em Java. A plataforma JACK permite que os programadores desenvolvam componentes que são necessários aos agentes BDI, além de oferecer alta performance em tempo de execução.

JADEX (JADEX, 2005)[4]. Jadex é uma plataforma de desenvolvimento orientado a agentes, onde os agentes são escritos em Java e XML. O Jadex segue o modelo BDI e pode ser usado em conjunto com diferentes plataformas de agentes.

2.5 Aprendizado de Máquina

Aprendizado de Máquina (*Machine Learning*) (AM) é uma área da Inteligência Artificial que permite dar aos computadores a habilidade de aprender sem que sejam explicitamente programados para isso (MICHIE *et al.*, 1994). O campo do aprendizado de máquina é concebido pela questão de como construir programas que automaticamente melhoram com a sua experiência.

Esta área inclui uma vasta gama de metodologias e algoritmos de aprendizagem, que podem ser classificados segundo diversos parâmetros, como a estratégia de aprendizagem subjacente, a representação do conhecimento e o domínio de aplicação.

Existem basicamente três tipos de Aprendizado de Máquina, são eles: Aprendizado Supervisionado, Aprendizado Não Supervisionado e Aprendizado por Reforço (RUSSELL; NORVIG, 2016). De acordo com Russell e Norvig (2016), os três principais tipos de aprendizagem podem ser determinados de acordo com os tipos de *feedback* recebido pelo algoritmo.

Segundo Russell e Norvig (2016):

- **Aprendizado não supervisionado**: O agente aprende padrões na entrada, embora não seja fornecido nenhum *feedback* explícito. A tarefa mais comum de aprendizagem não supervisionada é o agrupamento: a detecção de grupos de exemplos de entrada potencialmente úteis. Por exemplo, um agente de táxi pode desenvolver gradualmente um conceito

[3] http://agent-software.com.au/products/jack/
[4] https://www.activecomponents.org/

de “dia de tráfego bom” e “dia de tráfego ruim” sem nunca ter sido rotulados exemplos de cada um deles.

- **Aprendizagem supervisionada**: O agente observa alguns exemplos de pares de entrada e saída, e aprende uma função que faz o mapeamento da entrada para a saída.
- **Aprendizado por reforço**: O agente aprende a partir de uma série de reforços — recompensas ou punições. Por exemplo, a falta de gorjeta ao final de uma refeição em um restaurante dá ao agente do garçom a indicação de que algo saiu errado. Os dois pontos de vitória no final de um jogo de xadrez informam ao agente que ele fez a coisa certa. Cabe ao agente decidir qual das ações anteriores ao reforço foram as maiores responsáveis por isso.

Em certos domínios, como em problemas onde a interação é um aspecto central, ou em ambientes dinâmicos, a aprendizagem supervisionada ou não supervisionada pode não ser aplicável, pela impossibilidade de obter exemplos de treino convenientes. Nesses casos, a aprendizagem por reforço pode permitir ao agente a capacidade de adquirir um conhecimento do ambiente que não estava disponível em tempo de projeto.

2.5.1 Aprendizado por Reforço

Aprendizado por Reforço RL é uma técnica de aprendizagem em que o agente ou sistema deve aprender a selecionar as ações disponíveis, que alteram o estado do ambiente e utilizam uma recompensa (reforço) para definir a qualidade da sequência de ações (MITCHELL, 1997).

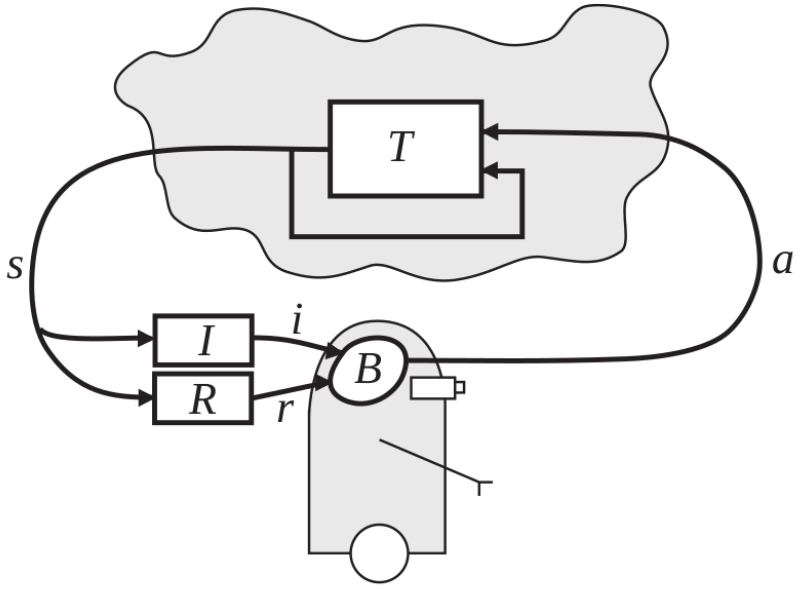
Segundo Guelpeli *et al.* (2003), o agente atua em um ambiente descrito por um conjunto de estados e pode executar, para cada estado, uma ação dentro de um conjunto de ações aplicáveis, recebendo um valor de reforço a cada vez que executa uma ação. Esse reforço indica o valor imediato da transição estado-ação-novo estado. O objetivo do agente é aprender uma política que maximize uma soma esperada desses reforços a longo prazo (GUELPELI *et al.*, 2003).

No aprendizado por reforço, o algoritmo escolhe uma ação em resposta a cada ponto de dados. O aprendizado por reforço é comum na robótica, em que o conjunto de leituras de um sensor, em um ponto no tempo, é um ponto de dados que o algoritmo deve usar para escolher a próxima ação do robô.

Formalmente, AR utiliza uma estrutura composta de estados, ações e recompensas

conforme mostra a Figura 4.

Figura 4 - Modelo padrão de Aprendizagem por Reforço



Fonte: (KAELBLING *et al.*, 1996)

No esquema demonstrado na Figura 4, o agente atua em um ambiente, T, descrito por um conjunto de possíveis estados. Em cada etapa de interação, o agente recebe como entrada, I, alguma indicação do estado atual do ambiente. Para cada estado é selecionada uma ação, a, dentro de um conjunto de ações possíveis, recebendo um valor de reforço, R, a cada vez que executa uma ação.

De acordo com Kaelbling *et al.* (1996), o comportamento do agente B deve ser escolher ações que tendem a aumentar a soma de valores de longo prazo do sinal de reforço. O agente pode aprender a fazer isso ao longo do tempo por tentativa e erro.

Em geral, o aprendizado por reforço está interessado em qualquer tipo de agente que precise aprender a escolher ações que alterem o estado de seu ambiente e onde uma função de recompensa cumulativa seja usada para definir a qualidade de qualquer sequência de ação (MITCHELL, 1997). O agente não precisa ter conhecimento das ações que deve tomar, mas deve descobrir quais ações o levam a maximizar a quantidade de recompensas recebidas.

2.5.2 Q-Learning

O algoritmo *Q-Learning*, proposto por Watkins e Dayan (1992), é um dos métodos de resolução do problema de aprendizagem por reforço. Essa técnica aprende a *qualidade* de

uma combinação estado-ação: $Q : S \times A \rightarrow R$, utilizando uma tabela (*Q-Table*) que mapeia qual o valor (qualidade) de uma ação em um determinado estado.

A função $Q(s,a)$ de recompensa futura esperada, ao se escolher a ação a no estado s, é aprendida por meio de um processo de exploração dos estados (tentativa e erro), onde a tabela *Q-Table* é atualizada seguindo a equação 2.1:

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha \times [r_t + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)] \tag{2.1}$$

Onde pode-se definir:

- *Q(s, a)* é a *Q-Function*;
- s é o estado
- a é ação
- α é a taxa de aprendizagem, $(0 \leq \alpha \leq 1)$;
- r é a recompensa recebida;
- γ é o fator de desconto, $(0 \leq \gamma \leq 1)$.

A taxa de aprendizado (α), determina em que medida as informações recém adquiridas substituirão as informações antigas. O valor de $\alpha = 0$ faz com que agente não aprenda nada, fazendo com que o algoritmo torne-se exclusivamente exploratório. Por outro lado, $\alpha = 1$ faz com que o agente considere apenas as informações mais recentes, ignorando o conhecimento prévio adquirido.

O fator de desconto (γ) permite ao agente determinar a importância de recompensas futuras. A ideia básica é pensar nas recompensas de maneira diferente ao longo do tempo e modelar a ideia de que as experiências anteriores são mais relevantes que as posteriores. Por exemplo, quando uma criança aprende a andar, as recompensas e punições são bem altas. Mas quanto mais envelhecemos, menos devemos precisar ajustar nosso comportamento de caminhar e as recompensas devem diminuir. Pode-se modelar isso incluindo um fator que diminui o valor da recompensa com o tempo.

Existem muitas maneiras para escolher ações baseadas nas estimativas aprendidas. O algoritmo *Q-Learning* utiliza uma política ε*-greedy* (ou, quase-guloso) para selecionar uma ação. Essa política pode selecionar a ação mais bem estimada até o momento ou selecionar uma ação de forma aleatória com pequena probabilidade ε, onde $0 \leq \varepsilon \leq 1$ indica qual a porcentagem de chance da ação ser selecionada de forma aleatória. Em geral, deseja-se uma política que tenha alguma aleatoriedade para promover a exploração do espaço de estados.

A forma procedural do algoritmo *Q-Learning* é retratada no Algoritmo 1.

Algorithm 1 Q-Learning

Para cada (s,a) inicialize $Q(s,a)$;
Observe s;
while o critério de parada não for satisfeito **do**
  Selecione a ação a usando a polıtica de ações ε-*greedy*;
  Execute a ação a;
  Receba a recompensa imediata $r(s,a)$;
  Observe o novo estado s' ;
  Atualize o item $Q(s,a)$ de acordo com a equação (2.1) ;
  $s \leftarrow s'$;
end while
FIM; =0

Fonte: (OTTONI *et al.*, 2013)

Resumidamente, pode-se enumerar os passos mais importantes do algoritmo *Q-learning* da seguinte maneira:

1. Inicialize a tabela de valores Q , $Q(s,a)$.
2. Observe o estado atual, s .
3. Escolha uma ação, a , para esse estado s, com base em uma política ε-*greedy*.
4. Execute a ação a e observe a recompensa, r , bem como o novo estado, s'.
5. Atualize o valor Q para o estado usando a recompensa observada e a recompensa máxima possível para o próximo estado. A atualização é feita de acordo com o fórmula 2.1 e os parâmetros descritos acima.
6. Defina um novo estado e repita o processo até que uma condição de parada seja satisfeita.

3 TRABALHOS RELACIONADOS

Nessa seção são detalhados alguns trabalhos relacionados que também propuseram utilizar SMA e/ou Aprendizado por Reforço para automatização de AmI, apresentando um breve resumo e os pontos de cada trabalho que serviram de base para a construção do trabalho proposto.

3.1 Uma Abordagem com Sistemas Multiagentes para Controle Autônomo de Casas Inteligentes

Em Andrade *et al.* (2016), foi proposta a construção de um sistema de controle autônomo que utiliza o paradigma de Sistemas Multiagentes e que explora os conceitos de capacidade de reação e adaptabilidade do sistema. Para a concepção do sistema foi proposta a utilização de uma plataforma de desenvolvimento de Sistemas Multiagentes, o *framework* JaCaMo, a elaboração de um cenário com situações a serem atendidas pelo sistema, o desenvolvimento da arquitetura e a implementação e teste em um ambiente simulado.

Em seu trabalho, Andrade *et al.* (2016) define como cenário para aplicação uma residência com três habitantes, dois adultos e uma criança, onde cada um deles tem suas preferências e atividades em que o sistema deve auxiliar. Além das preferências para cada habitante e os respectivos dispositivos eletrônicos, também foram elaborados dois casos para que se pudesse verificar a capacidade de adaptação do sistema a mudanças no ambiente em tempo de execução.

O primeiro caso elaborado por Andrade *et al.* (2016) aborda a necessidade do sistema de se adaptar a uma pessoa com deficiência auditiva, que necessita que a TV ative a função *closed caption*. O segundo caso retrata a mudança de hábito de um habitante que, para um certo horário do dia da semana, prefere um outro canal ao invés dos canais inicialmente projetados.

O trabalho em questão atinge seus objetivos de demonstrar ser possível construir soluções para casas inteligentes utilizando o paradigma de Sistemas Multiagentes proporcionado pela plataforma JaCaMo, obtendo um bom funcionamento no contexto de casas inteligentes e níveis de detalhamentos dos artefatos próximo a realidade.

O presente trabalho é uma extensão do trabalho aqui descrito, ou seja, a continuação de um trabalho futuro indicado pelos autores. A arquitetura e o sistema desenvolvido por Andrade *et al.* (2016) são utilizados e adaptados no presente trabalho.

Assim como o trabalho realizado em Andrade *et al.* (2016), o atual trabalho aborda

a utilização do paradigma de Sistemas Multiagentes para controle e adaptação de ambientes inteligentes. Entretanto, o trabalho relacionado utiliza artefatos de inteligência e aprendizagem de maneira simulada para alteração, inclusão e exclusão de planos dos agentes, enquanto que o presente trabalho busca utilizar algoritmos de Aprendizado por Reforço para gerar e/ou alterar planos de ação em tempo de execução baseado em experiências passadas.

3.2 *Action prediction in Smart Home based on Reinforcement Learning*

Em Hassan e Atieh (2014), foi apresentada a arquitetura de uma *Smart Home* que se adapta dinamicamente as preferências dos habitantes baseada em um processo contínuo de aprendizado sobre as mudanças ocorridas no ambiente.

O sistema proposto por Hassan e Atieh (2014) não faz suposições prévias sobre ações a serem automatizadas, mas monitora a interação do usuário com o ambiente e registra isso em planos de ações. Para implementação de adaptação, Hassan e Atieh (2014) desenvolvem um novo algoritmo de Aprendizado por Reforço denominado *Planning Q-learning Algorithm* (PQLA). O algoritmo desenvolvido modifica a execução do algoritmo *Q-learning*, mesclando-o com o conhecimento extraído dos planos registrados.

A arquitetura proposta possui mecanismos para observar as ações dos usuários em diferentes situações, contextos e ambientes com o objetivo de construir novos planos de ação em tempo de execução. O sistema aprende quando executar automaticamente as necessidades dos habitantes e, devido a utilização do algoritmo PQLA, as mudanças de comportamento dos habitantes também são identificadas.

O trabalho realizado por Hassan e Atieh (2014) assemelha-se bastante ao presente trabalho, no que diz respeito a criação de módulos de aprendizado para ambientes inteligentes. Entretanto, o presente trabalho propõe utilizar técnicas de aprendizado por reforço em conjunto com SMA, enquanto que no trabalho aqui descrito não são utilizados conceitos de agentes.

3.3 *Smart Home device usage Prediction using Pattern Matching and Reinforcement Learning*

Reaz *et al.* (2008) propõem um algoritmo que introduz uma nova técnica de previsão de ações do usuário utilizando uma combinação de correspondência de padrões e técnicas de aprendizado por reforço. Os autores utilizaram dados sintéticos para testar o algoritmo e o

resultado apresentado mostra que o algoritmo obteve um índice de 20% de melhoria em relação às técnicas disponíveis até aquele momento.

A técnica proposta por Reaz *et al.* (2008) envolve analisar o histórico de ações realizadas pelo usuário e aplicar métodos matemáticos para prever a próxima ação do usuário. O algoritmo tem dois componentes principais e são eles: o componente de aprendizado por reforço e o componente de correspondência de padrões.

Ao aplicar o método de aprendizado por reforço, o sistema recebe uma recompensa positiva por cada ação que é corretamente prevista ou uma recompensa negativa para cada ação errada. Usando este método, o sistema pode aprender e adaptar-se as ações ideais para cada usuário. A correspondência de padrões é usada para combinar a sequência mais recente de eventos do usuário com o histórico das sequências registradas, combinando os resultados obtidos de ambos os métodos.

Os trabalhos assemelham-se, pois ambos utilizam técnicas de aprendizado de máquina para implementar adaptabilidade para ambientes inteligentes e diferem-se porque os autores aplicam sua solução sobre dados sintéticos, enquanto este trabalho implementa um Sistema Multiagente e faz uso do algoritmo para alteração, inclusão e exclusão de planos de ação dos agentes que compõem o sistema.

3.4 Comparativo dos Trabalhos Relacionados

Abaixo, a Tabela 1 resume a análise dos trabalhos relacionados apresentados neste capítulo em comparativo com este trabalho.

Tabela 1 - Comparativo dos Trabalhos Relacionados

Trabalho	**Multiagente**	**Foco da Aplicação**	**Método de Aprendizado**	**Algoritmo**
Andrade *et al.* (2016)	SMA para resolução distribuída de problemas	Controle autônomo de Smart-Homes	Não se aplica	Não se aplica
Reaz *et al.* (2008)	Não utilizou	Predição de ações em uma Smart Home	*Pattern Matching and Reinforcement Learning*	*Q-Learning* e ONSI

Hassan e Atieh (2014)	Não utilizou	Predição de ações em uma Smart Home	*Reinforcement Learning*	*Planning Q-learning Algorithm*
Trabalho Proposto	SMA para resolução distribuída de problemas	Adaptação em Ambientes Inteligentes	*Reinforcement Learning*	Q-learning

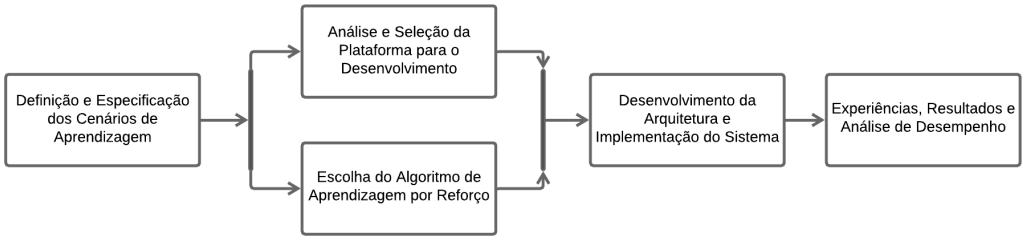
Fonte: Elaborada pelo autor.

4 IMPLEMENTAÇÃO E RESULTADOS

Neste capítulo são apresentados os passos da execução deste trabalho. Na Figura 5 pode ser visto o fluxo de trabalho que foi empregado. Inicialmente foi realizada a definição e especificação dos cenários de aprendizagem que o sistema deve suportar (Seção 4.1), logo após foi feita uma análise e seleção de uma plataforma para desenvolvimento de SMA (Seção 4.2) e a seleção de um algoritmo de RL (Seção 4.3), para que pudesse ser feita a implementação do sistema (Seção 4.4). Após isso foi feita uma análise dos resultados obtidos na implementação e nos cenários de teste de comportamento do sistema. (Seção 4.5).

Os códigos foram escritos utilizando a linguagem Java e *AgentSpeak* (BORDINI *et al.*, 2007). Todos os procedimentos e implementações aqui descritos estão disponíveis para reprodução[1].

Figura 5 - Passos Metodológicos



Fonte: Elaborada pelo autor

4.1 Definição e Especificação dos Cenários de Aprendizagem

A construção deste trabalho passa por criar um ambiente que possa representar as características de uma *Smart Home*, através da criação de modelo com a presença de sensores, atuadores e módulos de aprendizagem.

Como mencionado nos capítulos anteriores, o presente trabalho é uma extensão do trabalho de Andrade *et al.* (2016). A arquitetura e o sistema desenvolvido por Andrade *et al.* (2016) são utilizados e adaptados no presente trabalho. Em seu trabalho, Andrade *et al.* (2016) desenvolveu um artefato de aprendizado, que simula uma identificação de padrões de maneira randômica e fornece atributos para que os planos de um agente sejam atualizados em tempo de execução.

[1] https://github.com/fr-daniel/smart-home

A partir desse cenário, essa etapa foi utilizada para definir uma abordagem onde o sistema busca aprender a preferência de um habitante por serviço a ser ativado na residência que leva em conta o estado atual do ambiente e habitante.

Dentre as abordagens testadas em um primeiro momento, tentou-se definir as preferências do habitante para cada dispositivo específico em um determinado estado do ambiente, porém tal abordagem demostrou-se pouco eficiente visto que seria necessário um grande volume de treinamento do algoritmo de RL para cada dispositivo gerenciado pelo sistema em busca de se alcançar o comportamento desejado pelo habitante.

Uma nova abordagem foi estabelecida, onde o sistema busca aprender a preferência do habitante por um serviço a ser ativado no ambiente de acordo com a atividade que o habitante está realizando. Um serviço é composto pela definição das preferências do habitante para todos os dispositivos. Quando um serviço é selecionado, todos os dispositivos gerenciados pelo sistema devem alterar suas configurações para se adequar ao serviço selecionado. O habitante, por sua vez, devolve um *feedback* sobre o serviço selecionado para um determinado estado do ambiente.

A partir dessa abordagem, três elementos são essenciais para o desenvolvimento do sistema: os dispositivos e suas opções de configuração e possíveis valores, um conjunto de serviços e a configuração dos dispositivos para cada serviço, e a descrição dos estados do ambiente que são capturados por sensores.

4.1.1 Dispositivos

Um conjunto de sete tipos de dispositivos são controlados pelo sistema: TV, música, ar-condicionado, iluminação do ambiente, alarme, janelas e portas. A Tabela 2 mostra em detalhes a lista dos dispositivos e os possíveis valores das suas configurações.

Como é possível visualizar na Tabela 2, cada dispositivo possui características e funcionalidades próprias. A única característica em comum entre todos dispositivos é o seu estado, que determina se está ligado ou desligado para dispositivos eletrônicos ou aberto ou fechado para os dois dispositivos não eletrônicos. Os valores de cada configuração de cada dispositivo variam de acordo com o serviço cadastrado pelo habitante. O próximo tópico descreve a estrutura estabelecida para construção de um serviço.

Tabela 2 - Dispositivos Gerenciados pelo Sistema

Dispositivos	**Configurações**	**Valores**
Música	Estado	Ligado, Desligado
	Gênero	Pop, Funk, Rap, Rock, Gospel, Forró, Country, Eletrônica, Sertanejo, Jazz, Reggae, MPB
	Volume	Mudo, Baixo, Médio, Alto
TV	Estado	Ligado, Desligado
	Conteúdo	Netflix, Amazon Prime, Globo Play, HBO GO, TV por Assinatura
	Volume	Mudo, Baixo, Médio, Alto
Ar-condicionado	Estado	Ligado, Desligado
	Modo	Automático, Frio, Seco, Ventilador, Quente
Iluminação	Estado	Ligado, Desligado
	Cor	Azul, Cinza, Verde, Laranja, Vermelho, Amarelo
	Brilho	Brilhante, Médio, Escuro, Desligado
Alarme	Estado	Ligado, Desligado
Janela	Estado	Aberta, Fechada
Porta	Estado	Aberta, Fechada

Fonte: Elaborada pelo autor

4.1.2 Serviços

Como mencionado no inicio da seção, um serviço é composto por o conjunto de configurações para todos os dispositivos que o sistema pode gerenciar. O habitante possui a opção de cadastrar novos serviços no sistema. Para cada serviço a ser cadastrado as seguintes informações dos dispositivos são necessárias:

- **Música:** (’Estado’, ’Gênero musical’ e ’Volume’);
- **Tv:** (’Estado’, ’Conteúdo’e ’Volume’);
- **Ar-condicionado:** (’Estado’ e ’Modo’);
- **Iluminação:** (’Estado’, ’Cor’ e ’Brilho’);
- **Alarme:** (’Estado’);

- **Janela:** ('Estado');
- **Porta:** ('Estado').

Inicialmente, cinco serviços foram modelados para teste e desenvolvimento do sistema: Modo Ausente, Modo Dormir, Modo Assistir TV por Assinatura, Modo Estudar e Modo Escutar Música Eletrônica. Nas tabelas 3, 4, 5, 6 e 7 podem ser visualizadas as configurações selecionadas para os dispositivos em cada um dos serviços.

Tabela 3 - Serviço Modo Ausente

Dispositivos	**Configurações**
Música	(DESLIGADO, NENHUM, MUDO)
TV	(DESLIGADO, NENHUM, MUDO)
Ar-condicionado	(DESLIGADO, AUTOMÁTICO)
Iluminação	(DESLIGADO, PADRÃO, NENHUM)
Alarme	(LIGADO)
Janela	(FECHADO)
Porta	(FECHADO)

Fonte: Elaborada pelo autor

Tabela 4 - Serviço Modo Dormir

Dispositivos	**Configurações**
Música	(DESLIGADO, NENHUM, MUDO)
TV	(DESLIGADO, NENHUM, MUDO)
Ar-condicionado	(LIGADO, FRIO)
Iluminação	(LIGADO, PADRÃO, ESCURO)
Alarme	(LIGADO)
Janela	(FECHADO)
Porta	(FECHADO)

Fonte: Elaborada pelo autor

Tabela 5 - Serviço Modo Assistir TV por Assinatura

Dispositivos	**Configurações**
Música	(DESLIGADO, NENHUM, MUDO)
TV	(LIGADO, TV POR ASSINATURA, MÉDIO)
Ar-condicionado	(LIGADO, FRIO)
Iluminação	(LIGADO, AZUL, ESCURO)
Alarme	(LIGADO)
Janela	(FECHADO)
Porta	(FECHADO)

Fonte: Elaborada pelo autor

Tabela 6 - Serviço Modo Estudar

Dispositivos	**Configurações**
Música	(DESLIGADO, NENHUM, MUDO)
TV	(DESLIGADO, NENHUM, MUDO)
Ar-condicionado	(LIGADO, AUTOMÁTICO)
Iluminação	(LIGADO, PADRÃO, MÉDIO)
Alarme	(LIGADO)
Janela	(ABERTO)
Porta	(FECHADO)

Fonte: Elaborada pelo autor

Tabela 7 - Serviço Modo Escutar Eletrônica

Dispositivos	**Configurações**
Música	(LIGADO, ELETRÔNICA, ALTO)
TV	(DESLIGADO, NENHUM, MUDO)
Ar-condicionado	(LIGADO, VENTILADOR)
Iluminação	(LIGADO, VERMELHO, MÉDIO)
Alarme	(LIGADO)
Janela	(ABERTO)
Porta	(ABERTO)

Fonte: Elaborada pelo autor

De acordo com os valores das configurações recebidas, um arquivo com o formato *JavaScript Object Notation* (JSON) é gerado e armazenado no sistema. O código 1 exibe um exemplo do código JSON gerado para o serviço nomeado de "Modo Ausente".

Código-fonte 1: Representação em Json do Serviço Modo Ausente

```
"servico": {
  "nome": "ModoAusente",
  "musica": {"estado": "DESLIGADO", "genero": "NENHUM", "
     volume": "MUDO" },
  "tv": {"estado": "DESLIGADO", "conteudo": "NENHUM", "
     volume": "MUDO"},
  "ar_condicionado": {"estado": "DESLIGADO", "modo": "
     AUTOMATICO"},
  "ilumicacao": {"estado": "DESLIGADO", "cor": "PADRAO", "
     brilho": "NENHUM"},
  "alarme": {"estado": "LIGADO"},
  "janela": {"estado": "FECHADO"},
  "porta": {"estado": "FECHADO"},
}
```

Fonte: O Autor

4.1.3 Estado do Ambiente

O estado do ambiente é capturado por um conjunto de quatro sensores, definidos na Seção 4.4, que são responsáveis por detectar quais habitantes estão presentes na residência, além das alterações na atividade e localização desses habitantes.

O total de cinco elementos de contexto foram escolhidos para representar o estado do ambiente, sendo eles: identificador do habitante, localização, horário, dia da semana e atividade. Tais informações são úteis para a formulação de um estado que é utilizado pelo algoritmo de aprendizado na hora de selecionar um novo serviço. Os elementos de contexto são especificados por um conjunto de valores, conforme mostrado na Tabela 8.

Somente os locais que afetam a decisão do serviço foram adicionados no estado. Outros locais aumentam o espaço do estado e diminuem a velocidade de convergência da aprendizagem, sem contribuir para as funções do sistema. Essa consideração também foi aplicada na análise da identificação das atividades do habitante. Em nossa implementação, um conjunto de sete atividades são consideradas: "Deitado", "Sentado", "Andando", "Lendo", "Dormindo", "Comendo", "Assistindo"e "Nenhuma".

A possibilidade de detecção da atividade do habitante surge do trabalho desenvolvido por Khalili *et al.* (2009), onde estes desenvolvem métodos para capturar as informações sobre as atividades de um habitante em um cenário que utiliza dados reais. Como nosso artefato que monitora a atividade do usuário trabalha com informações pré-fixadas, ou seja, não faz uso de sensores reais, tais métodos aplicados por Khalili *et al.* (2009) não são utilizados.

As preferências do habitante para a qualidade de um serviço podem variar de tempos em tempos durante um dia. O usuário pode gostar de música eletrônica enquanto anda na sala de estar pela manhã de uma segunda-feira e pode preferir o silêncio enquanto lê um livro no horário da tarde de uma quarta-feira. Para discriminar essas situações, envolvemos intervalos de tempo e dia da semana como parte do estado.

Tabela 8 - Elementos de Contexto e Valores

Elementos de contexto	**Valores**
Habitante ID	Pai, Mãe, Filho, Filha
Dia da Semana	Segunda, Terça, Quarta, Quinta, Sexta, Sábado, Domingo
Horário	Manhã, Meio-dia, Tarde, Noite, Madrugada
Localização	Quarto, Sala de Estar, Cozinha, Corredor, Ausente
Atividade	Deitado, Sentado, Andando, Lendo, Dormindo, Comendo, Assistindo, Nenhuma

Fonte: Elaborada pelo autor

No método proposto, dividimos o dia em 5 intervalos de tempo sendo eles: "manhã", "meio-dia", "tarde", "noite"e "madrugada". Usar intervalos de tempo maiores tornaria nosso espaço de estado enorme. Nesse caso, seria difícil aprender o valor apropriado para cada par (estado, serviço) com número limitado de execuções, fugindo do objetivo deste trabalho.

Com base em todas as informações de contexto capturadas pelos sensores, um estado do tipo ***State*** é definido como uma como uma tupla de 5 elementos, conforme demostrado abaixo.

State (‘HABITANTE ID’, ’DIA DA SEMANA’, ’HORÁRIO’, LOCALIZAÇÃO, ’ATIVIDADE’)

Exemplos de algumas configurações de contexto para três estados diferentes são: E_1 = (PAI, SEGUNDA, NOITE, QUARTO, DORMINDO), E_2 = (MÃE, SÁBADO, MANHÃ, COZINHA, ANDANDO) e E_2 = (PAI, QUINTA, NOITE, SALA DE ESTAR, ASSISTINDO).

4.2 Análise e Seleção da Plataforma para o Desenvolvimento

Segundo Andrade *et al.* (2016), para o desenvolvimento do sistema multiagente que funcione da melhor maneira, em uma casa inteligente, é necessário utilizar uma plataforma de desenvolvimento que permita desenvolver uma abstração lógica do ambiente, que represente e seja a ponte entre as decisões dos agentes e o mundo real, permitindo detalhar o funcionamento dos objetos do mundo real e disponibilizar esses recursos aos agentes, tanto para entrada de dados quanto para atuação do sistema no ambiente (ANDRADE *et al.*, 2016).

Um conjunto de *frameworks* e plataformas têm sido desenvolvidos visando dar apoio ao desenvolvimento de SMA. De forma geral, estas plataformas estão associados a uma linguagem de programação para desenvolvimento dos agentes e fornecem um ambiente para a execução dos mesmos.

Para a realização deste trabalho foi necessária a seleção de uma plataforma que suporte a programação de agentes integrados em um SMA. Um estudo comparativo foi realizado, onde foram estabelecidos alguns critérios de escolha da plataforma de desenvolvimento.

As quatro plataformas de desenvolvimento de sistemas multiagentes mais utilizadas foram analisadas, a partir da bibliografia, para a seleção da ferramenta que melhor se compatibiliza ao contexto e aos objetivos do trabalho. Os critérios adotados para definir a plataforma a utilizada foram:

- Domínio primário de utilização da plataforma;
- Suporte a programação de agentes BDI;
- Existência de ferramentas para desenvolvimento;
- Linguagem de programação e Sistema Operacional;
- *Open Source*

De todos os *frameworks* analisados, apenas Jason e JADEX possuem suporte ao desenvolvimento de aplicações distribuídas compostas de entidades BDI autônomas, fornecem uma IDE gratuita com suporte a execução e depuração de código, podem ser executadas em diversos sistemas operacionais e são *Open source*.

Como já mencionado em seções anteriores, esse trabalho é uma extensão do projeto

desenvolvido por Andrade *et al.* (2016) e, apesar da plataforma Jadex ser aplicável para o desenvolvimento, este foi o principal fator que influenciou na seleção da plataforma Jason.

Na Tabela 9, é ilustrado um resumo das principais características que justificaram a escolha do *framework* Jason em detrimento das outras plataformas.

Tabela 9 - Comparativo dos *frameworks* de implementação

Características	***Frameworks***			
	Jadex	**JACK**	**Jade**	**Jason**
Domínio primário	Aplicações distribuídas compostas de entidades BDI autônomas	Ambientes dinâmicos e complexos	Aplicações distribuídas compostas de entidades autônomas	Aplicações distribuídas compostas de entidades BDI autônomas
Arquiteturas suportadas	BDI	BDI	Agentes reativos	BDI
Ferramenta de Desenvolvimento	IDE, execução e Debug e documentação	Ferramenta proprietária não gratuita	IDE, execução e Debug e documentação	IDE, execução e Debug
Linguagem	Java e XML	Java, *JACK Agent Language* (JAL) e XML	Java	Java e *AgentSpeak*
Sistemas Operacionais	Qualquer um com a Máquina virtual Java	Windows, Mac, Unix	Qualquer um com a Máquina virtual Java	Windows, MacOS e Linux
Open source	Sim	Não	Sim	Sim

Fonte: Elaborada pelo autor

As aplicações de Sistema Multiagente consistem não apenas de agentes, mas também de um ambiente no qual os agentes estão situados. Portanto, a construção de um SMA exige esforços em ambas as áreas, em que os aspectos ambientais não devem ser descartados.

Apesar do ambiente estar relacionado à existência física do agente, faz-se necessário estender esta noção para uma abstração computacional que mapeia as noções do mundo real

para o sistema multiagente. Isso permite o desenvolvimento rápido, por exemplo, de aplicativos de simulação, nos quais os agentes agem puramente nesses ambientes virtuais. Os ambientes virtuais também podem ser utilizados como forma de aprimoramento para um comportamento esperado em um ambiente real.

O *Common ARTifact infrastructure for AGents Open environments* (CArtAgO)[2] é uma estrutura de uso geral que possibilita programar e executar ambientes virtuais para sistemas com vários agentes.

O CArtAgO é baseado no metamodelo *Agents e Artifacts* (A&A) para modelar e projetar sistemas multiagentes. Os agentes são entidades computacionais executando algum tipo de atividade ou tarefa orientada a objetivos e os artefatos são os recursos e ferramentas dinamicamente construídos, usados e manipulados pelos agentes para apoiar a realização de suas atividades individuais e coletivas.

Portanto, com o CArtAgO, pode-se construir um modelo de programação simples para projetar e programar o ambiente computacional do agente, composto por conjuntos dinâmicos de artefatos de diferentes tipos.

Cada artefato é uma classe Java que estende através de herança a classe *Artifact* do CArtAgO. Cada artefato é composto de atributos e métodos próprios e herdados, e é através destes que ocorre o processo de interação com os agentes, executando código de baixo nível, atualizando a base de crenças dos agentes e interagindo com outros artefatos para propagar mudanças no ambiente.

A escolha do *framework* CArtAgO ocorreu por este possuir as características necessárias para o desenvolvimento de um ambiente virtual, além de oferecer amplo suporte as arquiteturas internas dos agentes desenvolvidos utilizando o *framework* Jason.

4.3 Escolha do Algoritmo de Aprendizagem por Reforço

O método de aprendizagem por reforço proposto neste trabalho tem como objetivo o ajuste dinâmico do serviço a ser ativado em uma *Smart Home* para um determinado estado do ambiente. O ajuste dinâmico de qual serviço deve ser ativado é importante para possibilitar a adaptação da casa as novas preferências dos seus habitantes.

Vários algoritmos foram desenvolvidos para implementar a RL. Para o nosso exemplo de algoritmo de aprendizado, implementaremos o *Q-Learning*. O algoritmo de aprendizado

[2] http://cartago.sourceforge.net/

Q-Learning, proposto por Watkins e Dayan (1992), foi desenvolvido pensando nos casos em que o agente não sabe como o mundo funciona e deve aprender como se comportar a partir da experiência direta com o mundo.

Em geral, existem duas abordagens para o aprendizado por reforço: (1) aprender um modelo do mundo a partir da experiência e, em seguida, usar o planejamento com esse modelo aprendido para ditar o comportamento do agente (baseado em modelo) e (2) aprender uma função de política ou valor diretamente da experiência (sem modelo). *Q-Learning* pertence a este último. O fato de ser livre de modelo favorece a aplicação do algoritmo neste trabalho, visto que a definição de cada estado do ambiente de uma residência e de cada ação aplicável para esse estado levaria a um enorme modelo e impossibilitaria o progresso do trabalho.

O *Q-Learning* demonstrou ser uma boa opção para vários tipos de aplicações que não podem realizar outros tipos de aprendizado de máquina e que tem como objetivo principal resolver um problema de adaptação, visto que o algoritmo possibilita aos agentes estarem em constante aprendizado.

O algoritmo *Q-Learning* é bastante popular na literatura por ser fácil de implementar e ser leve computacionalmente, além de ser uma opção já bastante utilizada quando deseja-se resolver problemas de ajuste dinâmico de preferências através da aprendizagem por reforço. Outro fator para a escolha do algoritmo é que toda a estrutura está disponível publicamente e são necessárias poucas modificações para adaptá-lo ao nosso problema.

4.4 Desenvolvimento da Arquitetura e Implementação do Sistema Multiagente

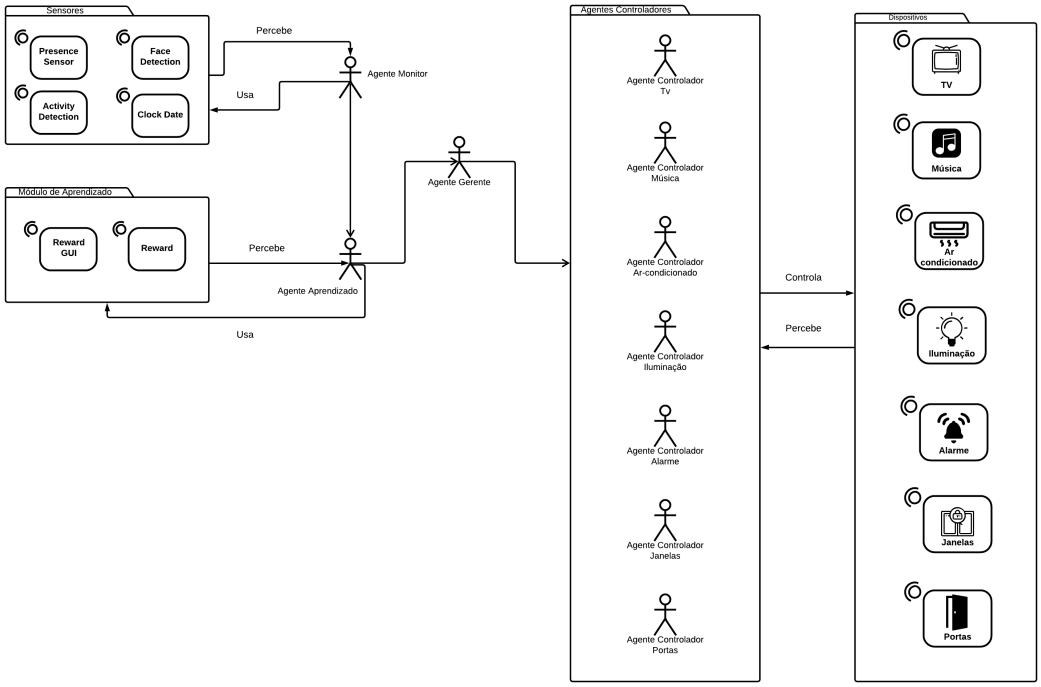
A construção do SMA e do ambiente de atuação do mesmo foi a principal etapa do trabalho. Nesta seção é apresentada a arquitetura proposta para o sistema, onde são detalhadas as características dos agentes e dos artefatos, e ilustrado como os agentes se comportam nos cenários de aprendizagem.

Na arquitetura proposta, assume-se que os agentes são executados sobre uma plataforma de agentes apresentada na seção 4.2, a qual dispõe de toda a infraestrutura e recursos necessários para desenvolvimento e execução do sistema.

Na Figura 6, temos o esboço da arquitetura do modelo desenvolvido neste trabalho para controle autônomo de uma Smart-Home. Como visto, a solução está dividida em camadas. Existe uma camada de ambiente, que representa uma abstração dos dispositivos a serem manipulados por uma camada de agentes, onde estes assumem os papéis de monitor, gerente,

controlador ou agente de aprendizado.

Figura 6 - Arquitetura do Sistema Multiagente



Fonte: Elaborada pelo autor

O fluxo dos dados no sistema acontece da seguinte forma. O Agente Monitor faz o sensoriamento e monta um estado do ambiente com os elementos de contexto definidos na Seção 4.1.3 e envia mensagens contendo o novo estado para o Agente de Aprendizado, utilizando um protocolo de comunicação disponível na linguagem Jason.

O Agente de Aprendizado processa o estado recebido através das mensagens e, por meio de ações internas, comunica-se com um código Java para o qual envia os dados que são utilizados pelo algoritmo de aprendizado. Posteriormente, um serviço é selecionado pelo Agente de Aprendizado que solicita ao habitante um *feedback* sobre esse serviço e, caso aprovado, envia o serviço selecionado para o Agente Gerente, que fica responsável por comunicar o serviço selecionado para todos os agentes controladores de dispositivos.

Todo o raciocínio é realizado entre os agentes através do processo de deliberação. As próximas seções detalham como foram desenvolvidos cada agente do sistema e quais artefatos cada agente manipula.

4.4.1 Detecção do Estado do Ambiente

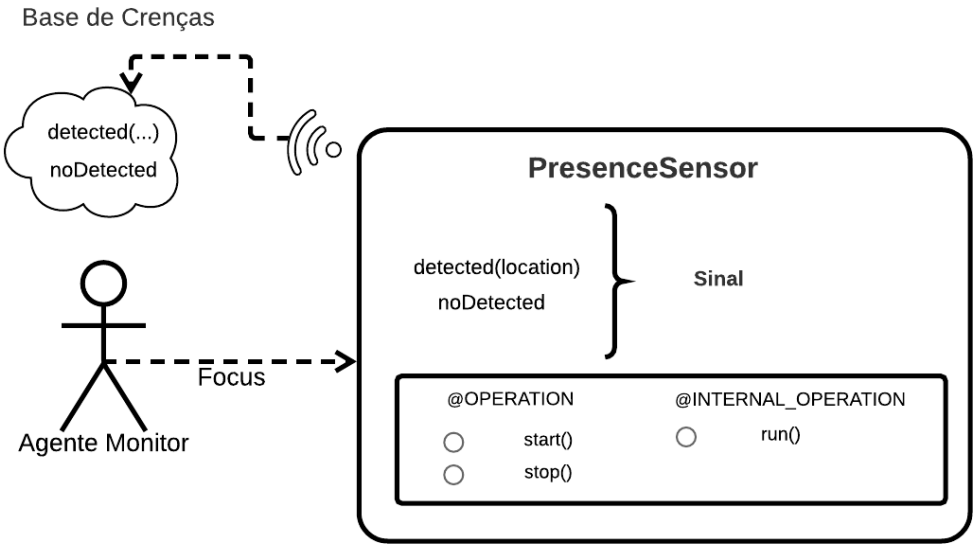
A detecção de alterações no ambiente é feita através do uso de um conjunto de sensores virtualmente implantados na residencia utilizando a plataforma CArtAgO. Cada sensor é responsável por capturar um dos elementos de contexto definidos na seção 4.1.3 para a formulação de um estado do ambiente.

Através desses sensores, é possível determinar quando o estado do ambiente mudou, permitindo que o agente monitor tenha sempre em mãos o estado mais atualizado do ambiente. Como demonstrado na Figura 6, a lista de sensores que compõem nossa solução são:

- ***Presence Sensor***: responsável por detectar um habitante e sua localização na residência;
- ***Face Detection***: responsável por detectar qual habitante está na residência (pai, mãe, filho, ...);
- ***Activity Detection***: responsável por detectar qual atividade o habitante está realizando;
- ***Clock Date***: responsável por capturar informações sobre o dia da semana e horário.

A interação entre o Agente Monitor e o artefato *Presence Sensor* é demostrada na Figura 7, onde o agente, ao receber sinais do artefato, atualiza sua base de crenças com essas novas informações. O mesmo processo ocorre com os demais sensores implementados.

Figura 7 - Processo de interação entre o Agente Monitor e um Sensor



Fonte: Elaborada pelo autor

Os planos possuem uma expressão condicional para serem executados, onde os dados recebidos dos sensores (artefatos) de entrada são testados nestas expressões. Quando o Agente Monitor recebe os dados de todos os sensores, um plano é ativado e a interação com o Agente de Aprendizado é iniciada.

Ao formar um novo estado do ambiente o Agente Monitor avisa ao Agente de Aprendizado que é necessário realizar a seleção de um novo serviço com base nesse novo estado, iniciando o processo demostrado na próxima seção.

4.4.2 Implementação do Algoritmo de Aprendizado por Reforço

O principal objetivo desta etapa foi definir os elementos necessários para a criação de um modelo de aprendizagem que utiliza a técnica de Aprendizado por Reforço e, posteriormente, implementar o algoritmo e sua integração com o agente de aprendizado.

O modelo de Aprendizado por Reforço proposto neste trabalho foi estruturado a partir da definição de um conjunto de ações (seleção de um serviço específico dentre um conjunto de serviços cadastrados), estados do ambiente e recompensas dadas pelo habitante para cada par (Estado, Ação).

A biblioteca de códigos java *Brown-UMBC Reinforcement Learning and Planning* (BURLAP)[3] destina-se ao uso e desenvolvimento de algoritmos de planejamento ou aprendizado com um ou vários agentes. BURLAP possui uma infraestrutura de classes Java altamente extensíveis e um conjunto de ferramentas de análise, com uma estrutura comum para a visualização de domínios e desempenho do algoritmo de aprendizado.

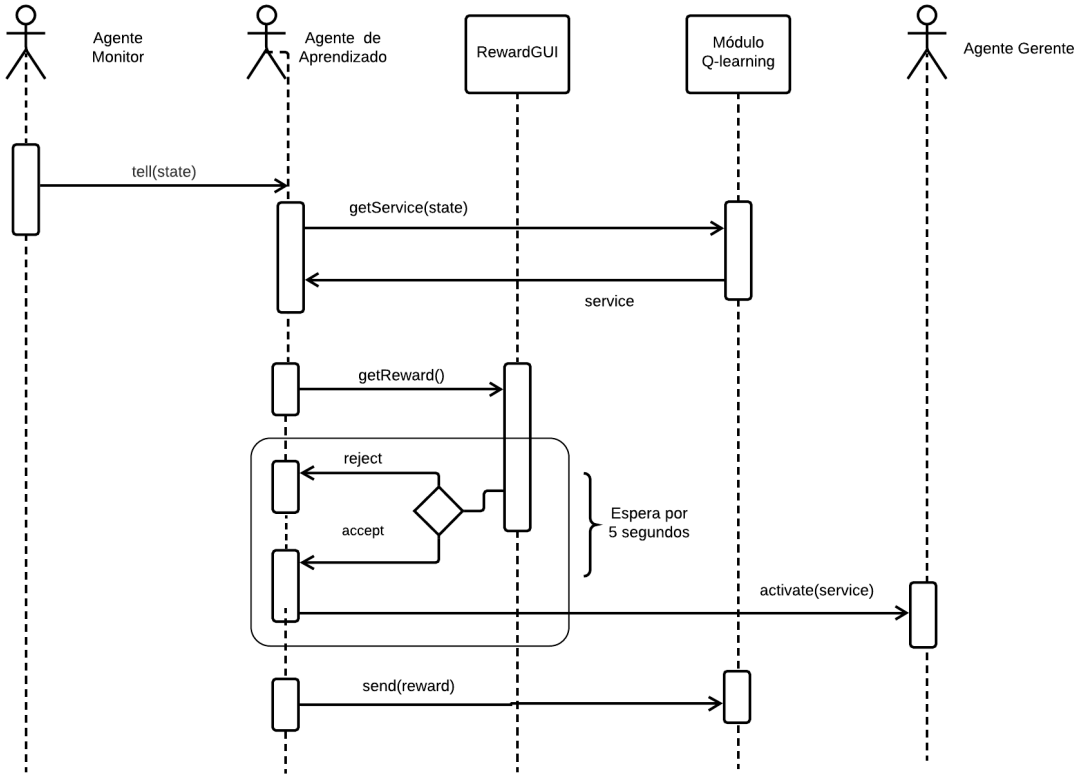
No âmbito do sistema, as classes definidas pela biblioteca BURLAP foram utilizadas para modelar o domínio do problema de RL utilizado pelo algoritmo *Q-learning* e também para desenvolvimento do próprio algoritmo.

A interação com o algoritmo *Q-learning* acontece através do Agente de Aprendizado. O processo de interação é demonstrado na Figura 8.

Conforme pode-se visualizar na Figura 8, após receber o estado atual do ambiente, por meio do agente monitor, o Agente de Aprendizado realiza o processo de solicitar ao módulo de aprendizado um serviço que leve em conta o estado recebido, após receber o serviço um *feedback* é solicitado ao habitante, utilizando um artefato do tipo *Graphical User Interface* (GUI). A Figura 9 ilustra o funcionamento desse artefato.

[3] http://burlap.cs.brown.edu/

Figura 8 - Interação entre o Agente de Aprendizado e o algoritmo Q-learning



Fonte: Elaborada pelo autor

O comportamento do artefato é aguardar por um período de 5 segundos um *feedback* do habitante, caso o habitante não se manifeste um *feedback* neutro é selecionado.

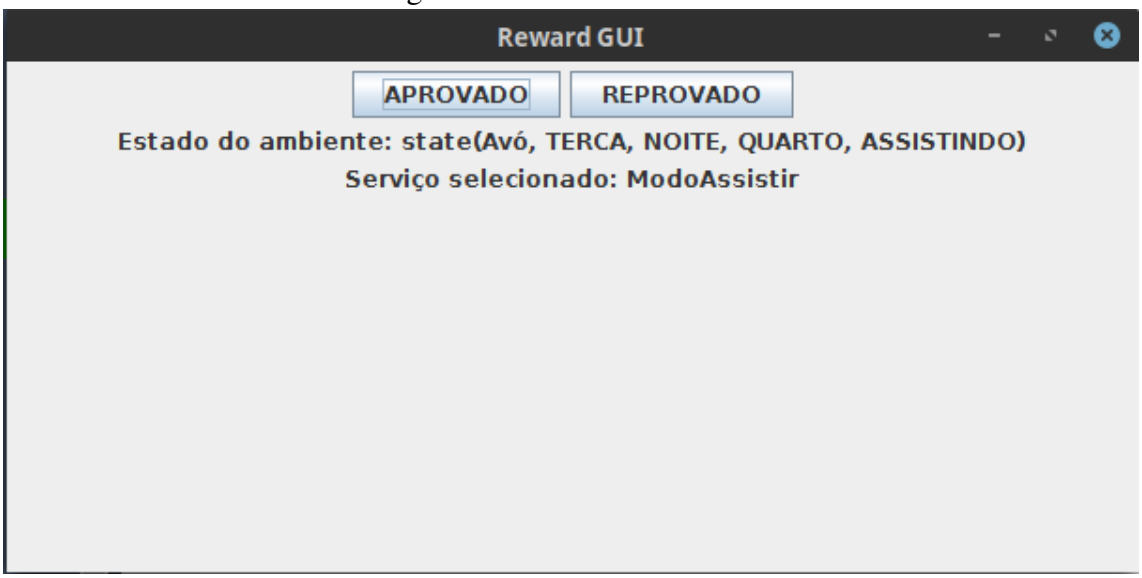
Um *feedback* positivo gera uma recompensa positiva para o algoritmo de aprendizado e permite que o serviço seja ativando na casa. Um *feedback* negativo ou neutro não autoriza a mudança do serviço e gera, respectivamente, uma recompensa negativa ou nula.

Os planos *p5*, *p6* e *p7*, presentes no código do agente de aprendizado e disponíveis no apêndice A, demostram o envio para o módulo de aprendizado, através da utilização do método interno do agente chamado *learning*, a recompensa para cada um dos casos mencionados. Como pode-se visualizar nos planos, apenas quando o agente observa que o serviço foi aprovado pelo habitante, é solicitado ao agente gerente que a ativação do serviço seja realizada, linha 34 do código disponível no apêndice A.

4.4.3 Ativação de um Novo Serviço

A atualização dos planos de cada agente controlador de dispositivo ocorre a partir do serviço recebido do agente gerente, onde um objeto java, que utiliza a estrutura do serviço armazenado em JSON, é instanciado e um método chamado *convertToPlan()* é utilizado para

Figura 9 - Artefato *Reward* GUI



Fonte: Elaborada pelo autor

converter as configurações de um serviço para planos.

As funções internas *add_plan* (adiciona novo plano) e *remove_plan* (remove plano), definidas pela linguagem Jason, são utilizadas para receber os planos já convertidos e realizar o processo de remoção e adição dos planos de cada agente controlador, conforme demostrado no Código Fonte 4.4.1.

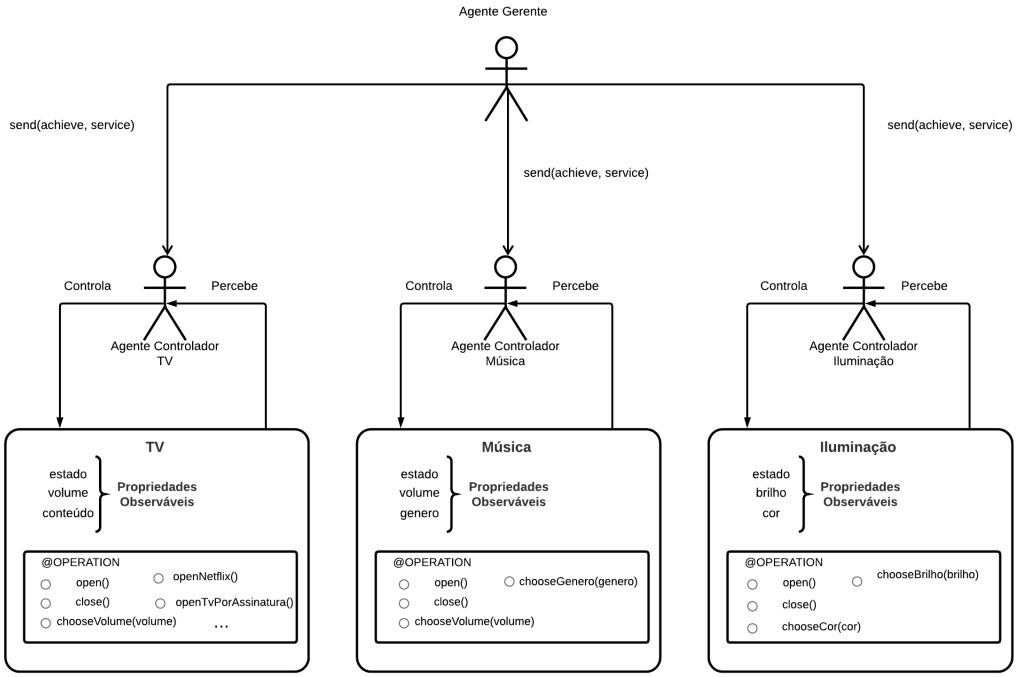
Listing 4.4.1 Atualização dos planos do Agente Controlador de TV

```
1  +!updatePlans(Service)
2      <-  cartago.invoke_obj(Service, getTvConfig, TvConfig)
3          cartago.invoke_obj(TvConfig, convertToPlan, Plans)
4          .remove_plan([selectBehavior, chooseVolume, chooseConteudo])
5          for ( .member(Plan, Plans) ) {
6              .add_plan(Plan);
7          };
8          !selectBehavior.
```

A Figura 10 representa o cenário onde ocorre a interação entre o Agente Gerente e alguns agentes controladores de dispositivos, quando o gerente recebe um novo serviço a ser ativado no ambiente. No cenário representado, o agente gerente recebe um serviço qualquer e solicita que os agentes controladores de TV, música e iluminação alterem as configurações dos dispositivos que cada um controla para a configuração estabelecida para o serviço;

Como ilustrado na Figura 10, cada agente controlador percebe as propriedades observáveis dos seus respectivos dispositivos para determinar o estado atual do dispositivo e usa

Figura 10 - Cenário de Ativação de um Novo Serviço



Fonte: Elaborada pelo autor

as operações implementadas nos dispositivos para alterar as configurações.

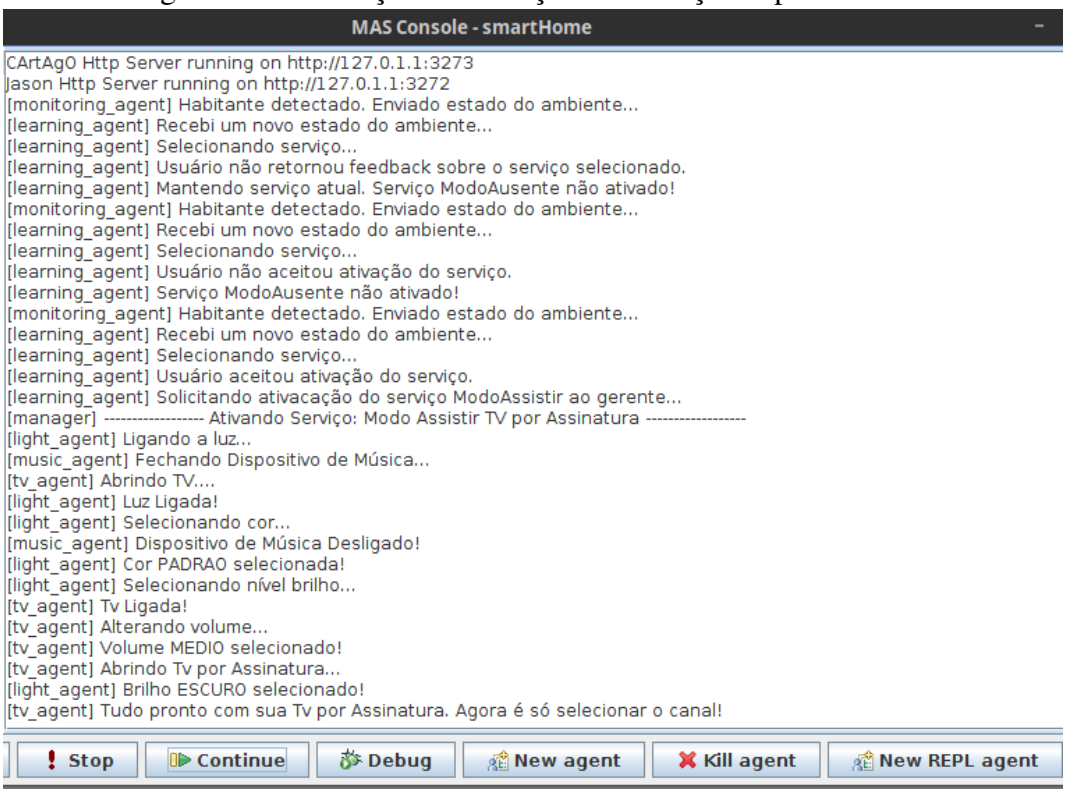
O processo de execução da ativação do serviço Modo Assistir TV por Assinatura é demostrada na Figura 11, onde, inicialmente, um habitante não fornece um *feedback* para o primeiro serviço selecionado pelo agente de aprendizado e não aceita a ativação do segundo serviço.

Quando o habitante aceita a solicitação do agente de aprendizado para a ativar o serviço Modo Assistir TV por Assinatura, o agente gerente notifica todos os agentes controladores. Em seguida, cada agente controlador realiza o processo de atualização de seus planos e passam a trabalhar na mudança das configurações dos seus dispositivos, conforme demostrado na Figura 11.

4.5 Experiências e Testes de Comportamento

Esta seção tem como objetivo apresentar de maneira simples o comportamento do sistema através de testes com o algoritmo *Q-learning* sobre um perfil de um habitante especifico. Entretanto, este não é um experimento de avaliação, tratando-se apenas da apresentação de como o algoritmo é utilizado para auxiliar no processo de ajuste dinâmico das preferências de um habitante.

Figura 11 - Simulação da Ativação do Serviço Tv por Assinatura



Fonte: Elaborada pelo autor

Antes de realizar os experimentos, precisamos definir um padrão de atividades a ser associado com um habitante da residência. A definição de um perfil para o habitante torna-se necessário para guiar uma política de recompensas para o algoritmo de aprendizado em uma primeira etapa de treinamento. O objetivo é observar se o algoritmo, após receber uma série de recompensas com base no perfil, consegue selecionar os serviços de acordo com as preferências do habitante.

Na inicialização do sistema, um treinamento não interativo é realizado a partir do perfil cadastrado para o habitante. Este perfil descreve as principais preferências que o habitante tem naquele momento. O agente *Q-learning* é treinado para maximizar as recompensas que levem aos estados definidos para o perfil do habitante.

Para o caso de teste, um perfil simplificado foi criado para um habitante que possui as seguintes preferências:

- Trabalha de segunda-feira a sexta-feira e deseja que o Serviço Modo Ausente seja ativado no período da manhã e tarde;
- De segunda-feira a sexta-feira o Serviço Assistir TV por Assinatura deve ser ativado sempre ao meio-dia;
- De segunda-feira a quarta-feira à noite, o Serviço Modo Estudar deve ser ativado;

- Às quintas e sextas-feiras à noite, o Serviço Modo Escutar Música Eletrônica deve ser ativado;
- De segunda-feira a sexta-feira durante a madrugada, o Serviço Modo Dormir deve ser ativado;
- Aos sábados, o serviço Modo Dormir deve ser ativado no período da noite e manhã. Durante o meio-dia, o Serviço Modo Escutar Música Eletrônica deve ser ativado. Já durante a tarde deve ser ativado o Serviço Assistir TV por Assinatura;
- O habitante está ausente durante todo o domingo e deseja que o modo ausente seja ativado.

Todos os experimentos foram realizados utilizando o algoritmo Q-Learning, com os seguintes parâmetros demostrados na Tabela 10.

Tabela 10 - Parâmetros de Simulação

Taxa de Aprendizagem (α)	0.3
Fator de desconto (γ),	0.9
Política de exploração ε-*greedy*	0.2

Fonte: Desenvolvida pelo Autor

Como o objetivo principal deste trabalho é demostrar a aplicabilidade de métodos de aprendizado em conjunto com SMA, não foram realizados demais testes sobre todos os possíveis valores para os parâmetros do algoritmo a fim de se verificar a influência de tais parâmetros, ficando essa etapa para um trabalho futuro. A escolha dos parâmetros foram determinados por meio de testes já realizados por outros autores em trabalhos semelhantes (KHALILI *et al.*, 2009; BERTON; BIANCHI, 2014; LUIZ *et al.*, 2006).

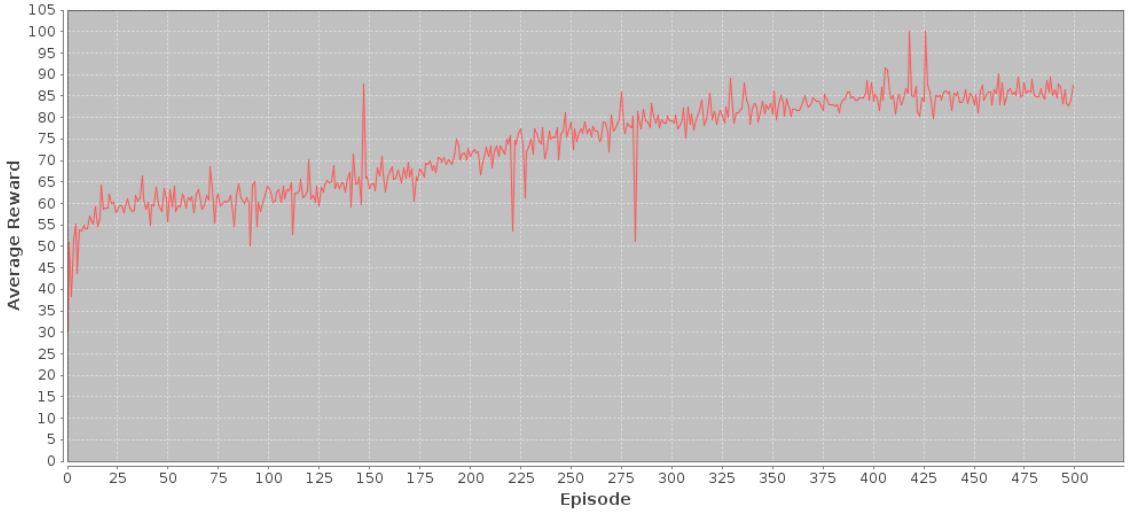
Em caso de alteração nas preferências do habitante, a taxa de aprendizagem desempenha uma papel considerável na adaptação. O valor 0.3 permite que uma quantidade expressiva de atualizações de valores ocorra na *Q-Table* em cada interação, fazendo com que o agente observe rapidamente as mudanças de comportamento do habitante. O valor do fator de desconto γ, estabelecido como 0.9 e próximo de 1, indica que as recompensas futuras têm um grau de importância elevado. O valor de ε-*greedy*, definido como 0.2, indica que em média 20% das ações poderão ser tomadas de maneira aleatória, ou seja, sem seguir o que foi aprendido até o momento.

Não é raro o habitante mudar de comportamento e recompensar o algoritmo de maneira negativa. A taxa de exploração ε-*greedy* define que o algoritmo deve sempre manter o "processo de exploração", escolhendo ações aleatórias para determinados estados. Essa política

aumenta a penalidade geral para um habitante que não altera suas preferências, mas evita que as mudanças de comportamento não sejam mais observadas e que o sistema faça uma convergência para uma decisão final não ideal.

A Figura 12 demonstra o comportamento do algoritmo *Q-learning* quando o as preferências do habitante são aquelas listas no inicio da seção, independentemente da localização e da atividade que o habitante estava fazendo. O eixo horizontal mostra a quantidade de episódios de teste e o eixo vertical mostra a recompensa média recebida em cada episódio.

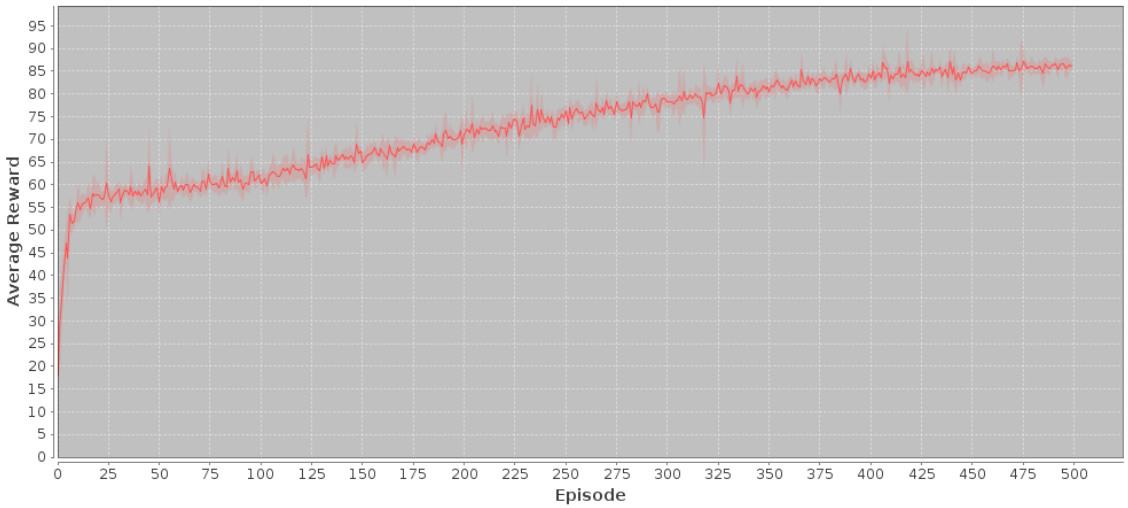
Figura 12 - Gráfico dos Reforços obtidos pelo algoritmo *Q-learning*



Fonte: Elaborada pelo autor

Os resultados da Figura 13 mostram as médias dos reforços obtidos durante a simulação com 500 episódios de teste. Como observado, *Q-Learning* consegue convergir para a uma recompensa próxima da recompensa máxima esperada.

Figura 13 - Gráfico dos Reforços médios obtidos pelo algoritmo *Q-learning*



Fonte: Elaborada pelo autor

A utilização da tabela *Q-Table* por parte do algoritmo *Q-learning* é um ponto negativo para problemas com uma grande variedade de estados, onde o custo computacional para gerar a tabela torna-se alto.

De maneira geral, o algoritmo *Q-learning* conseguiu melhorar e manter uma boa médias de recompensas positivas na configuração simplificada estabelecida para o perfil do habitante. Esse resultado se deve ao fato do total de estados possíveis nessa configuração ser menor do que o total de estados na configuração padrão, que leva em conta atividade e localização do habitante. A configuração simplificada estabelecida para os testes possibilitou que o agente avaliasse uma porcentagem maior de estados durante o treinamento quando comparado com a porcentagem de estados que haveriam de ser avaliados durante o treinamento na configuração padrão.

5 CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho apresentou o desenvolvimento (concepção, implementação e teste) de um Sistema Multiagentes que simula o comportamento de uma *Smart Homes* e busca adaptação a mudanças de comportamento dos habitantes utilizando um algoritmo baseado em Aprendizado por Reforço.

Para controle autônomo de *Smart Homes*, a abstração do mesmo, utilizando o paradigma de Sistemas Multiagentes proporcionado pela plataforma Jason e CArtAgO, mostrou-se bastante útil em ações relativas ao ambiente, demonstrando assim ser possível construir soluções para casas inteligentes utilizando Sistemas Multiagentes.

Já em relação a situações de adaptação às mudanças de comportamento do habitante, pôde ser observado que o *Q-learning*, implementado da forma tradicional, ou seja, usando uma simples tabela para inserir os valores obtidos pela função Q durante o processo de exploração por estados e ações, não é muito apropriado para o contexto de *Smart Homes*, visto que o custo computacional do algoritmo para mapear todos os estados e ações aplicáveis nesse tipo ambiente é muito grande.

Apesar disso, o método de adaptação ao usuário apresentado nesse trabalho apresenta algumas contribuições relevantes. A utilização de um algoritmo de aprendizado e sua integração com um SMA, demonstra a possibilidade de se trabalhar perfeitamente com agentes que utilizam técnicas de aprendizado em situações reais, sendo esta a principal contribuição deste trabalho.

Por fim, o desenvolvimento de um trabalho que propõe a utilização de métodos de adaptação é importante para possibilitar a criação novas abordagens que buscam aumentar a qualidade das relações dos usuários com os sistema computacionais. O código fonte deste trabalho está disponível no *github*[1] no endereço (https://github.com/fr-daniel/smart-home).

Uma limitação do trabalho apresentado é a ausência de tratamento de situações em que mais de um habitante esteja presente na residência. Embora o sistema consiga mudar as configurações dos dispositivos da residência com base no estado atual do ambiente e nas preferências do habitante, é necessário aplicar estes métodos em um cenário em que mais de um habitante esteja tentando utilizar os serviços da casa.

Além disso, esses experimentos podem ser realizados com a integração dos artefatos lógicos do CArtAgO com os dispositivos reais para se ter uma análise mais realista das funcionalidades do sistema.

[1] http://github.com/

Novas técnicas de Aprendizado por Reforço com otimizações com redes neurais, chamadas de *Deep Reinforcement Learning*, têm sido desenvolvidas para tornar os algoritmos de RL mais eficientes. Em um trabalho futuro, essas técnicas poderiam ser utilizadas para verificar se há alguma melhoria nos resultados do agente de aprendizado.

REFERÊNCIAS

AARTS, E.; WICHERT, R. Ambient intelligence. In: BULLINGER, H.-J. (Ed.). **Technology Guide: Principles – Applications – Trends**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 244–249. ISBN 978-3-540-88546-7.

ANDRADE, J. P. B.; OLIVEIRA, M.; GONÇALVES, E. J. T.; MAIA, M. E. F. Uma abordagem com sistemas multiagentes para controle autônomo de casas inteligentes. **XIII Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)**, 2016.

BERTON, P. A.; BIANCHI, A. **Aprendizado por reforço aplicado ao desenvolvimento de agentes humanoides no domínio do futebol de robôs simulado**. Centro Universitário da FEI, São Bernardo do Campo, 2014.

BORDINI, R. H.; BAZZAN, A. L.; JANNONE, R. de O.; BASSO, D. M.; VICARI, R. M.; LESSER, V. R. Agentspeak (xl): Efficient intention selection in bdi agents via decision-theoretic task scheduling. **Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3**, p. 1294–1302, 2002.

BORDINI, R. H.; HÜBNER, J. F.; WOOLDRIDGE, M. **Programming multi-agent systems in AgentSpeak using Jason**. [S.l.]: John Wiley & Sons, 2007. v. 8.

COOK, D. J.; AUGUSTO, J. C.; JAKKULA, V. R. Ambient intelligence: Technologies, applications, and opportunities. **Pervasive and Mobile Computing**, v. 5, n. 4, p. 277 – 298, [S.l], 2009. ISSN 1574-1192.

FRIEDEWALD, M.; COSTA, O. D.; PUNIE, Y.; ALAHUHTA, P.; HEINONEN, S. Perspectives of ambient intelligence in the home environment. **Telematics and informatics**, Elsevier, v. 22, n. 3, p. 221–238, 2005.

GAGGIOLI, A. Optimal experience in ambient intelligence. **Ambient intelligence**, Citeseer, v. 3543, n. 5, [S.l], 2005.

GIRARDI, R. Engenharia de software baseada em agentes. In: SN. **Procedimentos do IV Congresso Brasileiro de Ciência da Computação (CBComp 2004)**. [S.l.], 2004.

GUELPELI, M. V.; RIBEIRO, C. H.; OMAR, N. Utilização de aprendizagem por reforço para modelagem autônoma do aprendiz em um tutor inteligente. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. [S.l.: s.n.], 2003. v. 1, n. 1, p. 465–474.

GUERRA, C. A. N. **Um modelo para ambientes inteligentes baseado em serviços web semânticos**. Tese (Doutorado) — Universidade de São Paulo, 2007.

HASSAN, M.; ATIEH, M. Action prediction in smart home based on reinforcement learning. In: SPRINGER. **International Conference on Smart Homes and Health Telematics**. [S.l.], 2014. p. 207–212.

HÜBNER, J. F.; BORDINI, R. H.; VIEIRA, R. Introdução ao desenvolvimento de sistemas multiagentes com jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, [S.l], 2004.

JACK. **JACK Agent Language**. 2001. Disponível em: http://agent-software.com.au/products/jack/. Acesso em: 18 nov. 2019.

JADE. **Java Agent DEvelopment Framework**. 1999. Disponível em: https://jade.tilab.com/. Acesso em: 18 nov. 2019.

JADEX. **Jadex BDI Agent System**. 2005. Disponível em: https://www.activecomponents.org/. Acesso em: 18 nov. 2019.

JASON. **Java-based interpreter for an extended version of AgentSpeak**. 2007. Disponível em: http://jason.sourceforge.net/. Acesso em: 18 nov. 2019.

JUCHEM, M.; BASTOS, R. M. Engenharia de sistemas multiagentes: uma investigação sobre o estado da arte. **Relatório Técnico, Faculdade de Ciências Exatas da Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)**, 2001.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. **Journal of artificial intelligence research**, v. 4, p. 237–285, [S.l], 1996.

KHALILI, A.; WU, C.; AGHAJAN, H. Autonomous learning of user's preference of music and light services in smart home applications. **Behavior Monitoring and Interpretation Workshop at German AI Conf**, p. 12, [S.l], 2009.

LUIZ, A.; CELIBERTO, L.; BIANCHI, R. **Aprendizado por Reforço Acelerado por Heurística para um Sistema Multi-Agentes**. p. 1–10, [S.l: s.n], 2006.

MICHIE, D.; SPIEGELHALTER, D. J.; TAYLOR, C. *et al.* **Machine learning**. **Neural and Statistical Classification**, Technometrics, v. 13, [S.l], 1994.

MITCHELL, T. M. **Machine Learning**. 1. ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN 0070428077, 9780070428072.

OTTONI, A. L. C.; NEPOMUCENO, E. G.; OLIVEIRA, M. S. de; LAMPERTI, R. D. Análise do aprendizado por reforço aplicado a otimização em tomadas de decisões multiagente. In: BRAGA, A. d. P.; Bastos Filho, C. J. A. (Ed.). **Anais do 11 Congresso Brasileiro de Inteligência Computacional**. Porto de Galinhas, PE: SBIC, 2013. p. 1–6.

PEREIRA, L. M. Inteligência artificial: mito e ciência. **Revista Colóquio-Ciências**, v. 3, [S.l], 1988.

RAMOS, C.; AUGUSTO, J. C.; SHAPIRO, D. Ambient intelligence—the next step for artificial intelligence. **IEEE Intelligent Systems**, IEEE, v. 23, n. 2, p. 15–18, [S.l], 2008.

REAZ, M.; ASSIM, A.; IBRAHIMY, M.; CHONG, F.; MOHD-YASIN, F. Smart home device usage prediction using pattern matching and reinforcement learning. In: **Proceedings of the International Conference on System Identification and Control Problems**. [S.l.: s.n.], 2008. p. 28–31.

REIS, L. P. **Coordenação em Sistemas Multi-Agente**. Tese (Doutorado) — PhD thesis, Faculdade de Engenharia da Universidade do Porto, 2003.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence**: a modern approach. [S.l.]: Malaysia; Pearson Education Limited,, 2016.

SANTAELLA, L.; GALA, A.; POLICARPO, C.; GAZONI, R. Desvelando a internet das coisas. **Revista GEMInIS**, v. 4, n. 2, p. 19–32, [S.l], 2013.

SANTOS, B. P.; SILVA, L.; CELES, C.; BORGES, J. B.; NETO, B. S. P.; VIEIRA, M. A. M.; VIEIRA, L. F. M.; GOUSSEVSKAIA, O. N.; LOUREIRO, A. Internet das coisas: da teoria à prática. **Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos**, 2016.

SHADBOLT, N. From the editor in chief: ambient intelligence. **IEEE Intelligent Systems**, IEEE, n. 4, p. 2–3, [S.l], 2003.

SICHMAN, J. S. **Raciocínio social e organizacional em sistemas multiagentes**: avanços e perspectivas. Tese (Doutorado) — Universidade de São Paulo, 2003.

WATKINS, C. J.; DAYAN, P. Q-learning. **Machine learning**, Springer, v. 8, n. 3-4, p. 279–292, [S.l], 1992.

WEISS, G. **Multiagent systems**: a modern approach to distributed artificial intelligence. [S.l.]: MIT press, 1999.

WOOLDRIDGE, M. **Intelligent agents**. MIT Press London, Multiagent systems, v. 35, n. 4, p. 51, 1999.

WOOLDRIDGE, M. **An introduction to multiagent systems**. [S.l.]: John Wiley & Sons, 2009.

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent agents: theory and practice. **The Knowledge Engineering Review**, Cambridge University Press, v. 10, n. 2, p. 115–152, [S.l], 1995.

ZAMBIASI, S. P. *et al.* **Ambientes inteligentes**. Florianópolis, SC, 2002.

APÊNDICE A – CÓDIGO DO AGENTE DE APRENDIZADO

O código AgentSpeak(L) para o Agente de aprendizado é dado abaixo. No código, cada plano é anotado com um *label* (**p1**, **p2**, **p3**, ...) para que se possa referir ao plano no texto que segue. As ações que possuem um ponto ('.') em seu nome, denotam ações internas, uma noção introduzida em (BORDINI *et al.*, 2002). Essas ação são executadas internamente pelo agente, e não afetam o ambiente como as ações básicas que o agente executa.

A crença inicial do agente é sobre o identificador o último *feedback* retornado pelo habitante da residência, e que sua tarefa inicial é se iniciar, criando um artefato que recupera uma recompensa do habitante. Todos os planos são explicados abaixo.

O plano **p1** é usado quando o agente inicia sua atuação no sistema, ou seja, quando o agente é criado. Ele determina que o agente deve criar um artefato chamado *Reward* e deve executar a ação de *focus* sobre esse artefato.

O agente monitor provê a informação sobre a existência de um novo estado do ambiente. Quando o agente de aprendizado percebe que um novo estado foi recebido do agente monitor, a crença *state(Usuario, Dia, Horario, Localizacao, Atividade)* é adicionada à base de crenças de tal maneira que o plano **p2** pode ser então usado. O plano **p2** cria um objeto *State* e atualiza a crença do agente sobre o último *feedback* recebido do habitante. Posteriormente, **p2** determina que o novo objetivo do agente é selecionar um novo serviço.

O plano **p3** é usado para que o agente selecione um novo serviço a ser ativado na casa. O agente precisa utilizar um método interno *selectService(State, Service)* para enviar um estado para o algoritmo *Q-learning* e receber um serviço que é armazenado na variável *Service*, para posterior atualização da sua base de crença com a nova crença *rfOutputService*.

Com base no estado do ambiente e serviço selecionado, o plano **p4** é utilizado para que o agente solicite uma recompensa ao habitante utilizando um artefato do tipo GUI. O artefato do tipo GUI gera três possíveis novas adições de crença na base de crença do agente.

Quando o habitante clica no botão aprovado, uma nova crença *approved(Id)* é adicionada na base de crença do agente e o plano **p5** é ativado. O plano **p5** é utilizado para enviar uma recompensa positiva para o algoritmo *Q-learning* e para comunicar ao agente gerente que um novo serviço deve ser ativado.

Os planos **p6** e **p7** são utilizados para enviar uma recompensa negativa ou neutra para o algoritmo *Q-learning*, quando, respectivamente, uma crença *disapproved(Id)* ou *noReward(Id)* forem adicionadas a base de crença do agente.

O plano **p8** é utilizado para recuperar, da base de crença do agente, a informação sobre o último *feedback* retornado pelo habitante.

```
1  /* Initial beliefs and rules */
2
3  lastFeedbackId(0).
4
5  /* Initial goals */
6
7  !start.
8
9  /* Plans */
10
11 +!start                                                                    (p1)
12     <-  makeArtifact("Reward","smartHome.Reward", [], Reward);
13         focus(Reward).
14
15 +state(Usuario, Dia, Horario, Localizacao, Atividade)                      (p2)
16     <-  .print("Recebi um novo estado do ambiente...");
17         .concat("state(", Usuario, ", ", Dia, ", ", Horario, ", ", Localizacao, ",
18         "," Atividade, ")", State);
19         ?lastFeedbackId(N);
20         Id = N + 1;
21         -+lastFeedbackId(Id);
22         +rfInputState(Id, State);
23         -state(Usuario, _, _, _, _);
24         !selectService(Id, State).
25
26 +!selectService(Id, State)                                                 (p3)
27     <-  .print("Selecionando serviço...");
28         .wait(3000);
29         jia.selectService(State, Service);
30         +rfOutputService(Id, Service);
31         .wait(2000);
32         !reward(Id, State, Service).
33
34 +!reward(Id, State, Service)                                               (p4)
35     <-  .print(Id, Service, State);
36         .concat("RewardGUI", Id, Name);
37         makeArtifact(Name, "smartHome.RewardGUI", [], RewardGUI);
38         lookupArtifact("Reward", Reward);
39         linkArtifacts(RewardGUI, "out-1", Reward);
40         .term2string(Id, IdStr);
41         getReward(IdStr, State, Service)[artifact_name(Name)].
42
43 +approved(Id) : rfInputState(Id, State) & rfOutputService(Id, Service)    (p5)
44     <-  .print("Usuário aceitou ativação do serviço.");
```

```
45          jia.learning(Id, State, Service, 100);
46          .print("Solicitando ativacação do serviço ", Service, " ao gerente...");.
47          .send(manager, achieve, providerService(Service)).
48
49  +disapproved(Id) : rfInputState(Id, State) & rfOutputService(Id, Service)     (p6)
50      <-  .print("Usuário não aceitou ativação do serviço.");
51          jia.learning(Id, State, Service, -10);
52          .print("Serviço ", Service, " não ativado!").
53
54  +noReward(Id) : rfInputState(Id, State) & rfOutputService(Id, Service)       (p7)
55      <-  .print("Usuário não retornou feedback sobre o serviço selecionado.");
56          jia.learning(Id, State, Service, 0)
57          .print("Mantendo serviço atual. Serviço ", Service, " não ativado!").
58
59  +?lastFeedbackId(N)                                                          (p8)
60      <-  lastFeedbackId(P);
61          N = P.
62
```