# Universidade Fernando Pessoa

# Assistência à autonomia no domicílio com integração de Automação e Inteligência Artificial


Alessandro Miguez

Faculdade de Ciências e Tecnologia

Universidade Fernando Pessoa

Uma dissertação submetida ao grau de

*Mestre*

Orientadores: Prof Dr. José Torres e Prof. Dr. Pedro Sobral
Outubro de 2017

# Resumo

O *Ambient Assisted Living* (AAL) cresce em importância na investigação e investimentos públicos e privados, principalmente pelo envelhecimento da população mundial. Neste contexto, um problema que se busca resolver é a identificação e representação de atividades do cotidiano e para isto a Representação do Conhecimento mostra-se útil. A oferta de novas tecnologias *Internet of Things* (IoT) permitiu a criação de diversos protocolos de comunicação sem o suporte de um padrão, dificultando a integração de dispositivos em um único ambiente. Uma forma de solucionar este problema se dá na forma de aplicações de automação que traduzem estes protocolos em uma abstração que permita a interação com os dispositivos de modo uniforme. Esta disseminação de dispositivos IoT também disponibilizou uma quantidade massiva de dados para serem tratados, justificando o interesse de técnicas de Aprendizagem Máquina.

Este trabalho especifica, implementa e valida um sistema de ambiente inteligente que agrega automação e *Inteligência Artificial* (IA). A especificação dá-se na forma de uma arquitetura de referência, composta por três módulos, cujas tarefas são automatizar e uniformizar a recolha de dados, relacionar e dar significado a estes dados e aprender com estes. São propostos três casos de uso, que representam as tarefas desempenhadas por cada um dos módulos, tendo sido formulados de modo a demonstrar uma gradual integração destes componentes do sistema. A implementação trata das aplicações e técnicas utilizadas para o desenvolvimento de um protótipo funcional deste sistema. A validação é realizada com a avaliação dos testes realizados para cada um dos casos de uso propostos.

É apresentada uma discussão sobre temas relacionados ao trabalho, especificamente sobre aplicações de automação, Representação do Conhecimento, suas diferentes linguagens, Aprendizagem Máquina e são analisados trabalhos e artigos destas áreas. O resultado do trabalho é a especificação de uma arquitetura de referência para um sistema de acompanhamento de idosos que moram sozinhos e apresentados três casos de uso que representam as tarefas de cada um dos módulos.

# Abstract

*Ambient Assisted Living* (AAL) grows in importance in scientific researches and investment by companies and governments, mainly due to the aging of the world population. In this context, a problem that is sought to solve is the identification and representation of activities of daily living. For this purpose the Knowledge Representation proves itself useful. The offer of new technologies known as *Internet of Things* (IoT) result in the creation of several communication protocols, making difficult to integrate devices from different vendors into a single environment. One way to solve this problem is the use of a middleware layer that integrates these protocols into an abstraction and allows interaction with devices in a uniform manner. This dissemination of IoT devices also triggered a large quantity of data to be processed, justifying the use of *Machine Learning* (ML) techniques. This work specifies, implements, and validates a smart environment system that aggregates Automation and *Artificial Intelligence* (AI). The specification presents a reference architecture, composed by three modules, whose tasks are to automate and standardize the collection of data, to relate and give meaning to that data and to learn from it. Three use cases are proposed and represent the tasks performed by each of the modules, have been formulated in such a way as to demonstrate a gradual integration of these components of the system. The implementation addresses the applications and techniques used for the development of a functional prototype of this system. The validation is performed with the evaluation of the tests performed for each of the proposed use cases. A discussion on work-related topics is presented, specifically on automation applications, Knowledge Representation, their different languages, ML and are analyzed papers and articles of these areas.

The result of the work is the specification of a reference architecture for a monitoring system for elderly people who live alone and are presented three use cases that represent the tasks of each of the modules.

# Agradecimentos

Aos meus orientadores, Professores Doutores José Torres e Pedro Sobral, cujas críticas contribuíram para o aperfeiçoamento deste documento. Aos demais professores do curso pela disponibilidade em auxiliar.

Aos meus pais pelo amor e suporte incondicionais.

Ao meu irmão, sempre presente, apesar da distância.

Aos meus amigos, presentes ou não, pelas palavras e gestos de apoio e carinho.

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Acrónimos

**AAL** *Ambient Assisted Living*

**ADL** *Activities of Daily Living*

**AI** *Artificial Intelligence*

**API** *Application Programming Interface*

**DAML9** *DARPA Agent Markup Language*

**DL** *Description Logics*

**ECG** *Eletrocardiograma*

**FOL** *First-Order Logic*

**HRMI** *Heart Rate Monitor Interface*

**HTTP** *Hiper-Text Transfer Protocol*

**IA** *Inteligência Artificial*

**IoT** *Internet of Things*

**JSON** *JavaScript Object Notation*

**KIF** *Knowledge Interchange Format*

**KB** *Knowledge Base*

**ML** *Machine Learning*

**OSGi** *Open Services Gateway Initiative*

**RDF** *Resource Description Framework*

**RDFS** *RDF Schema*

**RFID** *Radio-Frequency Identification*

**OIL** *Ontology Interchange Language*

**ONU** *Organização das Nações Unidas*

**OO** *Orientado a Objetos*

**OWL** *Web Ontology Language*

**PLC** *Power Line Communication*

**REST** *Representational State Transfer*

**SGBD** *Sistema de Gestão de Bases de Dados*

**SWRL** *Semantic Web Rules Language*

**TIC** *Tecnologias de Informação e Comunicação*

**UE** *União Européia*

**URL** *Uniform Resource Locator*

**URI** *Uniform Resource Identifier*

**XSD** *XML Schema Definition*

**W3C** *World Wide Web Consortium*

# Capítulo 1

# Introdução

Mark Weiser (Weiser, 1991) trata da importância do que ele chamava desaparecimento da tecnologia no plano de fundo, ou seja, de como é importante que o uso de uma tecnologia pelos utilizadores se torne habitual, quase como uma segunda natureza.
Este desaparecimento é um dos requisitos necessários para o que ele chama computação ubíqua. Outro requisito seria a saturação dos espaços com dispositivos eletrónicos dotados de capacidade de computação e comunicação. Assim, a Computação Ubíqua possui um papel de destaque no âmbito dos espaços inteligentes, visto que diminui ou elimina completamente as fronteiras entre o "real" e o digital. A diminuição desta fronteira abre novas possibilidades de interações, bem como possibilita a automatização dos espaços, libertando as pessoas de tarefas tediosas de modo a poderem focar esforços para atividades de mais alto nível ou dando-lhes tempo, por exemplo, para o lazer e convívio familiar ou social.
Todavia, se em 1991 o custo dos dispositivos eletrónicos era uma restrição importante, atualmente tal não mais se verifica, ao menos no mundo desenvolvido, de modo que o termo *Internet of Things* (IoT) já é algo relativamente disseminado e familiar, mesmo para o público não especializado.
A Domótica, enquanto domínio da automação doméstica, tem-se desenvolvido rapidamente com o aparecimento de novos dispositivos inteligentes, sensores, câmeras, eletrodomésticos. A criação e utilização de novas tecnologias e protocolos de comunicação fazem com que estes aparelhos sejam cada vez menores, mais baratos e mais econômicos. Entretanto, há ainda a necessidade de alguma interação dos utilizadores com estes sistemas, obviamente, para a implantação e configuração, tarefas que são dificultadas pela multiplicidade e disparidade das tecnologias e protocolos anteriormente referidos.
Contudo, a atuação humana também é exigida para a definição dos eventos a serem verificados pelos sistemas e as consequências desta ocorrência. Ocorre que a condição a ser verificada para a atuação do sistema quase sempre decorre da consulta a entradas decorrentes de um conjunto de sensores, relativamente a dados brutos gerados por estes dispositivos. Dados brutos, ou seja, sem terem sido objeto de processamento. Entende-se

aqui a tarefa de processamento de dados segundo a teoria da caixa preta, segundo a qual uma quantidade de dados de entrada, aqui considerados como os dados brutos, alimentam uma caixa preta, que possui um algoritmo de processamento de dados, resultando em dados de saída, ou dados processados.
Uma das formas mais simples de processar estes dados busca determinar qual deve ser o seu resultado conforme uma lista de regras deterministicamente estabelecidas. Estes dados brutos são analisados com pouco interesse no contexto em que estão inseridos. Outra forma de processamento destes dados pode ser a inclusão destes numa dimensão semântica, ou seja, fazer com que a ocorrência de um dado evento possua algum significado. Para cumprir esta tarefa, lança-se mão de técnicas de representação do conhecimento, de modo que eventos de mais alto nível podem ser derivados de conjuntos de dados brutos, conforme regras estabelecidas previamente. Os dados brutos ou eventos passam a ser incluídos em um contexto mais amplo e podem relacionar-se com outros eventos.
Um ambiente saturado de dispositivos torna-se capaz de gerar dados de naturezas diversas e em grande escala. Uma das formas de processamento de dados visa tirar partido desta abundância de dados e encontrar padrões que não estão evidentes à primeira vista. A emergência destes padrões requer a utilização de cálculos estatísticos para obter um modelo que pode ser utilizado para classificar ou prever resultados em novas instâncias de dados. Nestes termos, são utilizadas técnicas de Aprendizagem Máquina para esta forma de processamento de dados.

## 1.1 Enunciado do problema

O trabalho está inserido no contexto do IoT, mais especificamente do AAL, com a proposta de integrar, em um ambiente inteligente, tecnologias de automação e técnicas de IA, a saber, Representação do Conhecimento e Aprendizagem Máquina, possibilitando uma maior inter-relação entre dados oriundos de diversas origens, bem como lidar com a abundância de dados gerados por estas ferramentas, de modo a melhorar a qualidade de vida dos utilizadores deste ambiente.
Consideram-se origens de dados, dispositivos eletrônicos com capacidades de comunicação ou mesmo aplicações que comunicam com o sistema. Um dos problemas que este relacionamento de dados pode ajudar a resolver é o reconhecimento de atividades ditas complexas. Há diversas abordagens do problema, entre elas a de que uma atividade complexa é o conjunto de atividades atómicas ou eventos sucedidos ao longo do tempo. O modo como estes eventos se relacionam, bem como o significado que eles possuem é objeto das atividades da representação do conhecimento.
Outro problema relaciona-se com a manipulação de grandes volumes de dados e a oportunidade que eles representam em identificar padrões não evidentes. A Aprendizagem Máquina, enquanto conjunto de técnicas e algoritmos, em grande parte estatísticos, ajuda

a criar um modelo a partir destes dados e responder a perguntas que envolvam, por exemplo, classificação de instâncias ou predição de resultados.

## 1.2 Objetivos

O presente trabalho foi realizado com o objetivo de propor uma arquitetura de sistema e um protótipo que visam aliar as três formas de processamento de dados mencionadas anteriormente. Um módulo, para além das tarefas de automação que um sistema AAL deve possuir, atua como um agente reflexivo baseado em modelo. Outro módulo implementa um agente baseado em conhecimento com sua *Knowledge Base* (KB) própria e capacidade de inclusão e de derivar novas sentenças. O último módulo atua como agente de aprendizagem implementando algoritmos de classificação.
Há, atualmente, uma tendência em focar muito dos esforços de investigação no âmbito da utilização de técnicas de Aprendizagem Máquina para uma panóplia de contextos, como uma "bala de prata" da IA. Longe de diminuir a importância da Aprendizagem Máquina na resolução de problemas, o que se propõe no trabalho é a utilização, em conjunto, desta técnica com outras mais aptas a resolver problemas em determinados contextos.

## 1.3 Estrutura do trabalho

O trabalho é introduzido pelo Capítulo 1 que expõe as motivações do trabalho, o problema que se busca resolver e os objetivos que se pretende alcançar, bem como delineia os temas a serem tratados nos demais capítulos.
O Capítulo 2, apresenta a discussão sobre o AAL, visto que foi o contexto adotado, seguindo-se a uma análise do conceito de IoT, e qual a sua consequência em termos de diversidade de tecnologias. Realiza-se, ainda, uma exposição do conceito de *Inteligência Artificial* (IA), dos seus agentes e como estes concretizam as três formas de processamento de dados mencionados acima. Finaliza-se esta exposição com uma discussão sobre trabalhos relacionados a todos estes domínios e alguns que utilizaram técnicas semelhantes às aqui adotadas.
Após esta exposição teórica, no Capítulo 3 foi proposta a arquitetura do sistema, que é formado por três módulos, automação, representação do conhecimento e aprendizagem. A cada módulo foram atribuídas tarefas a serem executadas e o modo como serão realizadas. O capítulo finaliza com a proposta de três casos de uso, ordenados por grau de complexidade e de quantidade de módulos necessários para a sua realização.
Em seguida, no Capítulo 4, descreveu-se como o protótipo foi implementado. Na secção destinada ao módulo de automação é apresentada a aplicação de automação adotada, as suas mais-valias, o modo como foi parametrizada e algumas regras de automação desen-

volvidas. Sobre o módulo de representação do conhecimento, fez-se uma breve exposição da linguagem de ontologia *Web Ontology Language* (OWL), da linguagem de regras *Semantic Web Rules Language* (SWRL) e as aplicações utilizadas para a modelagem e manipulação de dados da ontologia desenvolvida.

O Capítulo 5 é dedicado à demonstração de validação do protótipo, segundo o modo como os casos de uso propostos foram cumpridos, nos diversos cenários possíveis para cada um deles, bem como avaliação dos resultados obtidos, seja na forma de mais valias que a arquitetura proporciona, seja nas desvantagens que possui.

Encerra-se o trabalho com o Capítulo 6, em que se discutiu os resultados obtidos, as limitações e fortalezas do sistema, bem como se aponta modificações a serem feitas no sistema e a proposição de trabalhos a serem desenvolvidos no futuro.

# Capítulo 2

# Estado da Arte

## 2.1 Introdução

Este capítulo apresenta alguns conceitos e uma revisão de literatura. Pode ser dividido em duas partes: uma onde é feita uma análise de teor prático ou de ferramentas e outra de cariz teórico. Na primeira, serão mencionadas algumas aplicações de automação de ambientes, discutidas as suas valias e justificada a escolha de uma delas com base em requisitos pré-definidos. A análise teórica tratará de temas afetos ao âmbito do *Ambient Assisted Living* (AAL) e da *Inteligência Artificial* (IA), com análise de artigos concernentes com estes temas. A secção de IA será dividida entre Representação do Conhecimento e Aprendizagem Máquina, sendo que no primeiro tópico serão mencionados os assuntos de lógica, ontologias e a representação do tempo.
Esta estrutura deve-se ao fato do trabalho discutir questões vinculadas a arquiteturas de sistemas AAL, que dependem de aplicações de automação de ambientes, bem como uma abordagem que combina técnicas de IA dedutivas (Representação do Conhecimento) e indutivas (Aprendizagem Máquina).

## 2.2 *Ambient Assisted Living* (AAL)

O AAL pode ser definido como o uso das *Tecnologias de Informação e Comunicação* (TIC) no cotidiano de uma pessoa ou ambiente de trabalho que lhe permita permanecer mais tempo ativa, socialmente conectado e vivendo de modo independente na velhice (Monekosso et al., 2015). A ideia é povoar os ambientes com sensores e atuadores de modo a melhorar a qualidade de vida dos seus habitantes.
A principal motivação da AAL decorre do constante aumento da população idosa no mundo inteiro. Um relatório da ONU[1] apresenta uma estimativa que no ano de 2050, pela primeira vez na história da humanidade, a população idosa, maiores que 65 anos de idade,

[1] http://www.un.org/en/development/desa/population/theme/ageing/WPA2015.shtml

irá superar a população mais jovem (Figura 2.1). Muitos são os fatores que explicam este fenómeno, especialmente o avanço tecnológico na área da saúde que aumentou a expectativa de vida ao redor do globo. Este cenário traz novos desafios sociais, económicos, culturais e uma procura maior por soluções, inclusive aquelas com suporte a tecnologias de informação. Um dos desafios que a AAL busca resolver passa por melhorar a qualidade de vida de pessoas com necessidades especiais, dentre as quais se destacam os idosos, nas suas residências, de modo a que possam desempenhar suas atividades cotidianas com algum grau de independência.

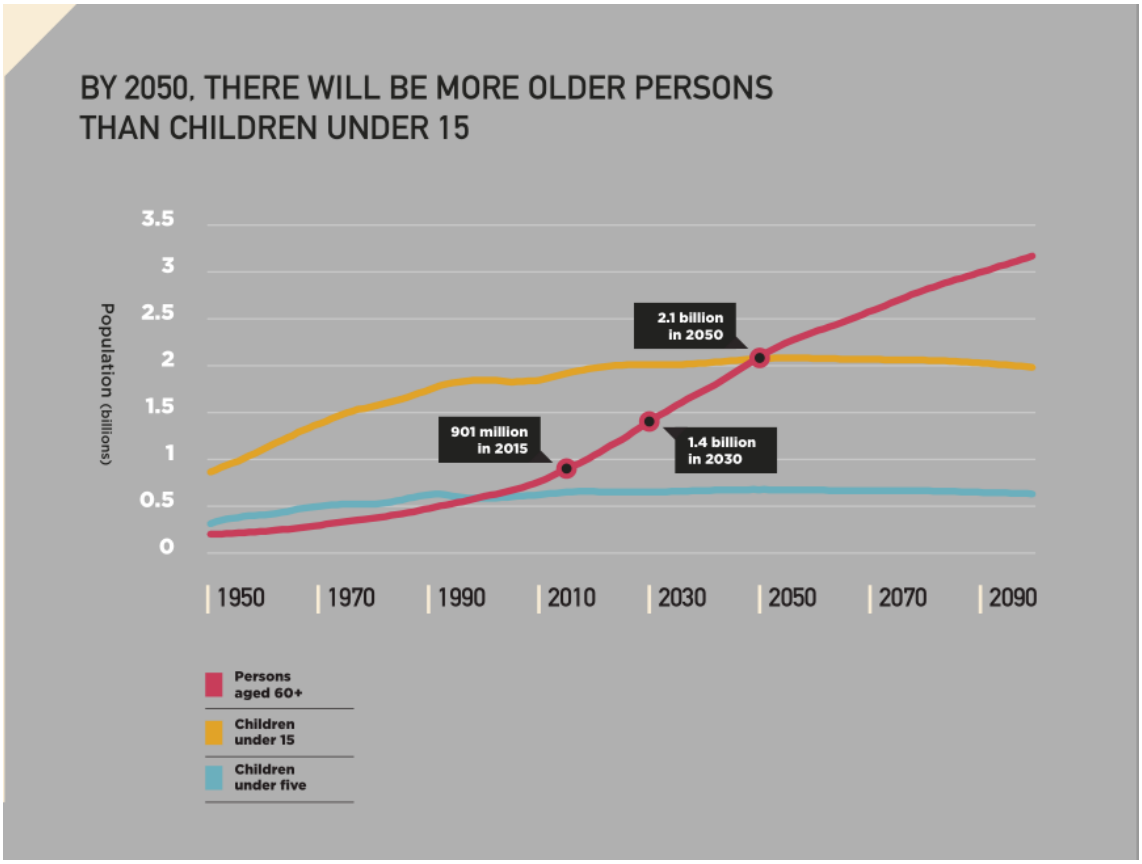



Figura 2.1: *Relatório da ONU sobre envelhecimento da população mundial (Nations, 2009)*

A importância do AAL pode ser demonstrada pelo empenho com que instituições governamentais tratam o tema, tendo como exemplo significativo o programa AAL, *Active and Assisted Living*, encabeçado pela *União Européia* (UE) e 19 países. Este programa é a nova denominação do projeto *Ambient Assisted Living Joint Programme*, criado no âmbito do Horizonte 2020[1], com um orçamento estimado em €700 milhões e destina-se a financiar projetos públicos e/ou privados que visem melhorar a qualidade de vida da

[1] `https://ec.europa.eu/programmes/horizon2020/`

população idosa, bem como incentivar a base industrial europeia[1].

A maior longevidade da população traz preocupações acerca de como garantir uma maior autonomia mesmo quando enfrentam situações comuns às pessoas desta idade, sejam elas da ordem da saúde, como doenças cardíacas, ósseas, ou mesmo mentais, ou de ordem sociais, como o facto de morarem sozinhos e afastados de seus familiares. É bem documentado que a ausência de atividades e interações sociais contribui para uma piora na qualidade de vida e pode contribuir para o aparecimento ou agravamento de condições psicológicas indesejadas, por exemplo, depressão. Portanto, ao falar-se de aumentar a autonomia ou garantir a independência dos habitantes de AAL's, quer-se, também, garantir que estas pessoas sejam capazes de manter ou criar vínculos sociais. O artigo (Li et al., 2015) traz um levantamento acerca de diversos projetos AAL, elencando objetivos de cada um deles, bem como tecnologias utilizadas. Continua o artigo categorizando os trabalhos em diferentes áreas, a saber, auxílio nas tarefas diárias, assistência na mobilidade, cuidados com saúde, reabilitação, inclusão social e comunicação.

A proximidade com a área da IA é quase evidente, visto que é preciso que haja agentes inteligentes para automatizar uma gama de atividades e monitorar parâmetros ambientais ou pessoais dos utilizadores. Muitos destes projetos envolvem o reconhecimento de atividades do cotidiano, com recurso a diversas técnicas de IA, sejam elas de teor indutivo ou dedutivo.

O artigo (Sendra et al., 2015) apresenta uma arquitetura comunicação inteligente para AAL, na qual utiliza técnicas de Aprendizagem Máquina para lidar com dados gerados de diferentes sensores implantados no ambiente e retirar informações como padrões de utilização de divisões, bem como ações de caráter mais determinísticos como ligar lâmpadas quando alguém entra em uma região pouco iluminadas. O trabalho em questão trata da existência de múltiplos protocolos de comunicação e utiliza técnicas de comunicação redes *ad hoc*, redes *mesh*, e discute a questão da otimização das trocas de informação entre os diferentes dispositivos no ambiente. O resultado é um sistema capaz de identificar hábitos dos utilizadores, determinar estados do ambiente e reagir a eles por meio de ações no próprio ambiente ou enviando alertas aos utilizadores ou aos seus cuidadores.

Em (Blasco et al., 2013) é apresentada uma cozinha inteligente adaptada às necessidades de pessoas com deficiências e idosos. Descreve o sistema e os seus diferentes módulos, adotando o padrão *Open Services Gateway Initiative* (OSGi) para implementação, define como princípios a capacidade de resistir à obsolescência dos dispositivos ou protocolos e habilidade para interoperar com sistemas AAL já existentes. A interoperação é garantida por um módulo chamado *e-Servant*, que é responsável por comunicar com os demais dispositivos. Cita como protocolos suportados, *Power Line Communication* (PLC), *Radio-Frequency Identification* (RFID), *ZigBee*, *Bluetooth* e *WiFi*.

[1] `http://www.aal-europe.eu/about/`

## 2.3 *Internet of Things* (IoT) e Aplicações de Automação

### 2.3.1 Introdução

O termo IoT é relativamente recente e ainda não possui uma definição consolidada. Contudo, será adotada a definição dada em (Rellermeyer et al., 2008), na qual considera o IoT como "a possibilidade de dotar todos os objetos do cotidiano com a habilidade de se identificarem, comunicar com outros objetos e possivelmente computar". Neste sentido, é de se notar que IoT contém o *Ambient Assisted Living* (AAL), nos termos em que este é uma especialização daquele. Esta secção tem como objetivo tratar do IoT como um conjunto de tecnologias de modo a realizar tarefas num dado contexto, no caso, o AAL.

A existência de objetos do cotidiano com capacidades de comunicação e computação permite, entre outras coisas, a possibilidade de interações de modo a facilitar a automação de tarefas, bem como a produção em massa de dados que anteriormente exigiriam demasiado esforço. Sem um termómetro com estas capacidades, as tarefas de obtenção e inserção, em uma base de dados, de amostras de temperatura de um determinado ambiente teriam de ser feitas manualmente, o que seria tedioso para o responsável e difícil de escalar. Assim, a adoção de tecnologias IoT traz como uma das consequências mais notáveis o conceito de *Big Data*, o aumento exponencial de dados disponíveis e passíveis de serem analisados para retirar informações com valor agregado.

A ausência de padronização no modo como os dispositivos se comunicam levou a uma diversidade de protocolos utilizados por dispositivos e softwares de IoT, dificultando e encarecendo a implantação de automação nos ambientes em geral. A automação doméstica sofre, em particular com isto, de modo a gerar uma procura por soluções que abstraiam os diferentes modos de comunicar dados numa rede.

As oportunidades de negócio neste segmento são óbvias e é possível verificar o investimento de empresas do porte de Google[1], Amazon[2] e Apple[3], para citar algumas, em produtos e serviços que se destinem a este mercado. A investigação científica, por outro lado, também têm dedicado esforços em aplicações deste género como ferramentas *ad hoc*, ou seja, cujo escopo se encerra no âmbito da própria investigação, de modo a contornar as dificuldades de comunicação apontadas.

Outra vertente de esforços vem das comunidades *open source* que unem esforços em torno de projetos que se propõe a difundir a automação com baixo custo de implantação. Deve-se mencionar que a proposta inicial destes projetos comunitários é difundir a utilização de tecnologias IoT e torná-las úteis para o contexto da domótica doméstica. O que cada um dos projetos implementa à sua própria maneira é eliminar a barreira da diversidade de protocolos. Assim sendo, estas aplicações transcendem o propósito original e podem ser

[1] https://developers.google.com/iot/
[2] https://aws.amazon.com/pt/iot/
[3] https://developer.apple.com/homekit/

consideradas úteis para contornar o obstáculo dos protocolos em projetos científicos que visem abordar o contexto de ambientes inteligentes, como é o caso do presente trabalho. A escolha por utilizar ferramentas derivadas desta abordagem deve-se à redução dos custos envolvidos, sejam eles financeiros ou de desenvolvimento. A escolha da ferramenta teve como critérios:

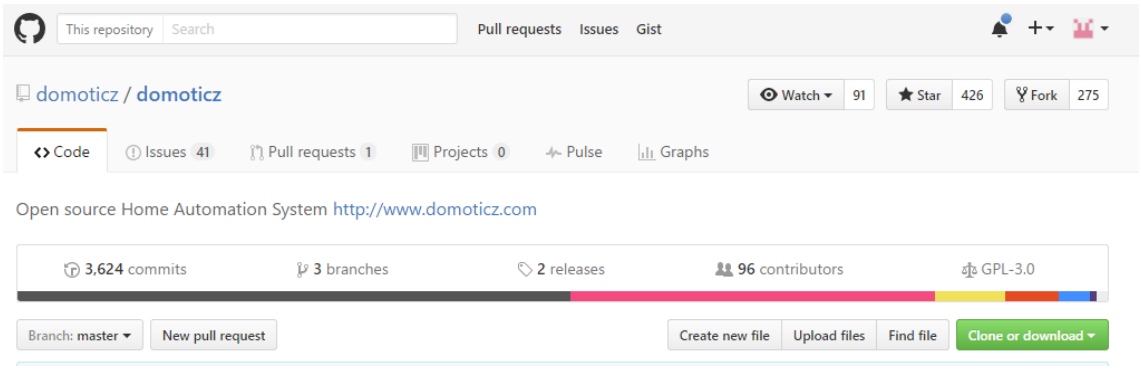
- participação da comunidade de desenvolvimento;
- clareza da documentação;
- suporte a múltiplos sistemas operativos;
- suporte ao maior número de protocolos, especialmente série e *Hiper-Text Transfer Protocol* (HTTP).

A seguir são apontadas algumas destas iniciativas, com breve descrição das abordagens, tecnologias suportadas e características principais.

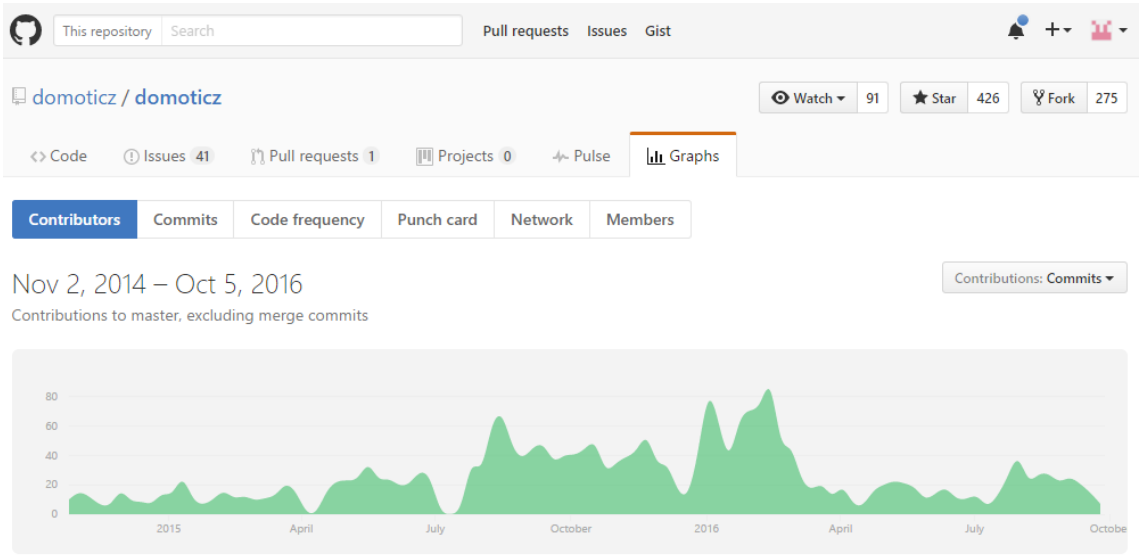
### 2.3.2 Aplicações de Automação

A aplicação (Domoticz, a) tem como propósito ser executada em dispositivos de baixo consumo, poder computacional e em diversos sistemas operativos. Foi implementada com recurso a linguagem C++ e destina-se a integrar diferentes sensores e atuadores, possuindo, em sua documentação[1], uma lista fechada de dispositivos suportados. Conta também com uma interface gráfica web para configuração e parametrização dos dispositivos. O processo de automação pode ser feito pela interface gráfica ou, programaticamente, na linguagem Lua[2].
As figuras 2.2 apresentam a atividade do repositório da aplicação ao longo da sua existência.
O projeto possui um manual em formato PDF, bem como um fórum e uma página wiki, todavia, o processo de parametrização da aplicação mostrou-se mais complexo que as alternativas, para além da falta de menção de interação por meio do protocolo HTTP.
A aplicação (Jeedom, a) foi desenvolvida em PHP apenas para o ambiente Unix, sendo recomendado o sistema operativo Debian.

[1] `https://www.domoticz.com/wiki/Hardware`
[2] `https://www.lua.org/`

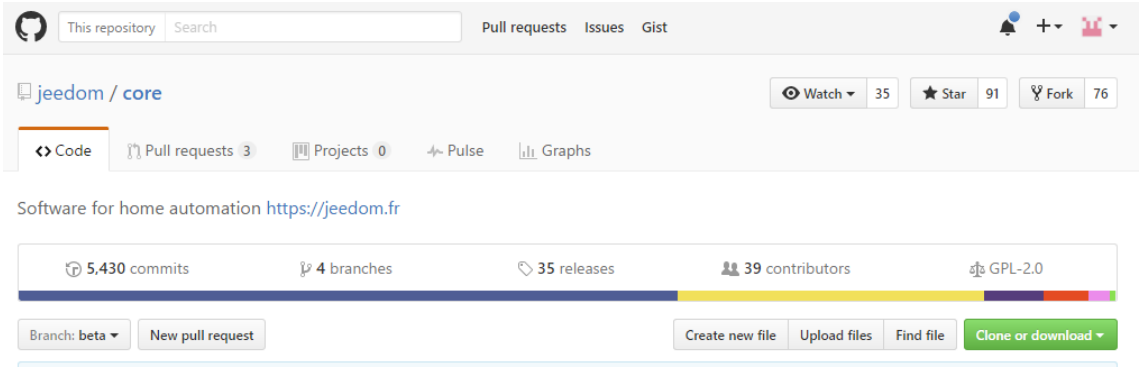



(a)

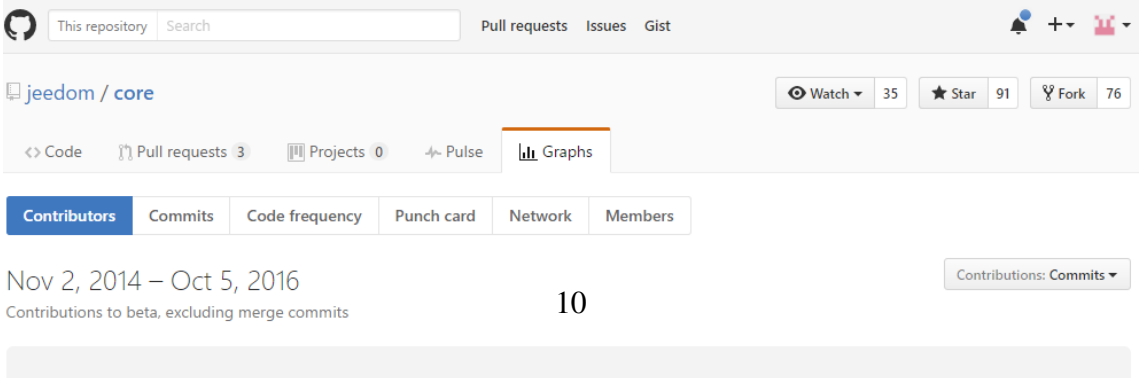



(b)

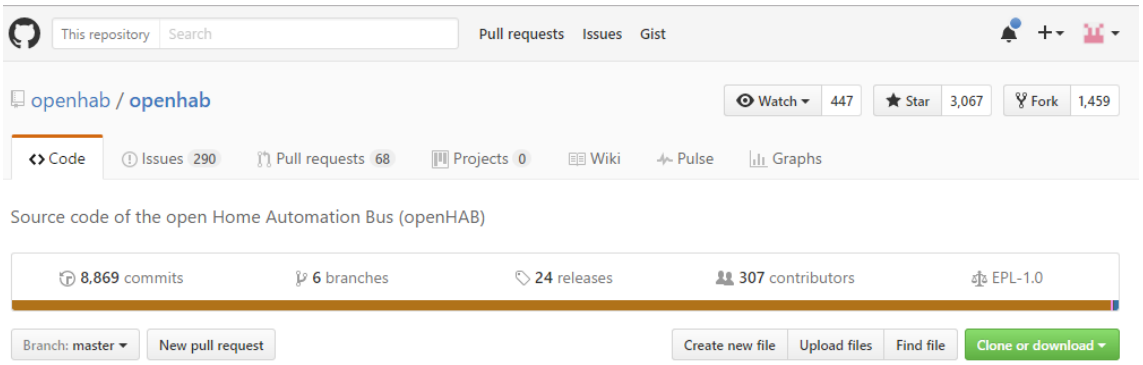
Figura 2.2: *Atividade do repositório Domoticz (Domoticz, b)*



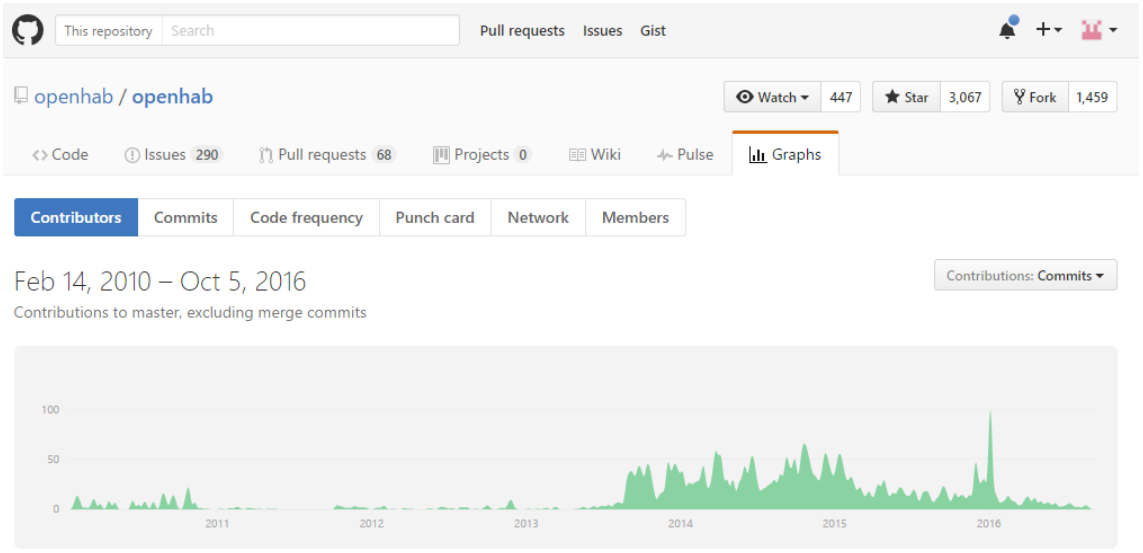

(a)



A aplicação (OpenHAB, a), desenvolvida em Java, traz uma abordagem distinta das demais aplicações já analisadas. Ao invés de suportar dispositivos específicos, propõe-se a suportar uma espécie de tradução dos protocolos para uma abstração criada pela *framework*. Desta maneira o desenvolvimento de mais protocolos pela comunidade implica em disponibilização de mais plugins na forma de addons, cujas inclusões e remoções não comprometem a execução do programa. O suporte a uma grande quantidade de protocolos, aliada à facilidade de configuração e parametrização dos dispositivos, tarefas que serão abordadas em capítulo posterior, bem como a existência de comunidade mais ativa que as demais, foram determinantes para a sua escolha.




(a)




(b)

Figura 2.4: *Atividade do repositório OpenHAB (OpenHAB, c)*

### 2.3.3 Conclusão

A tabela 2.1 abaixo representa o resumo da análise feita anteriormente, aí constando os critérios utilizados, a saber, ser ou não um projeto de código aberto, a participação da

comunidade, a qualidade atribuída à documentação, os protocolos/tecnologias suportados e o suporte ou não a múltiplas plataformas ou sistemas operativos. Esta tabela serviu de base para a escolha da ferramenta a ser adotada na implementação do projeto.

Tabela 2.1: *Comparação entre Aplicações de Automação*

| Aplicação | Open Source | Comunidade | Documentação | Protocolos | Multi-Plataforma |
|---|---|---|---|---|---|
| OpenHAB | ✓ | +++ | +++ | +++ | ✓ |
| Domoticz | ✓ | ++ | ++ | + | ✓ |
| Jeedom | ✓ | + | + | ++ | ✗ |

## 2.4 *Inteligência Artificial* (IA)

### 2.4.1 Introdução

*Inteligência Artificial* (IA) é o ramo da ciência da computação que estuda, dentre outros temas, os chamados agentes inteligentes, seja analisando o modo como realizam os processos internos de ilações, seja pelo modo como atuam no ambiente. Um agente é qualquer entidade capaz de perceber o ambiente a sua volta por meio de sensores e interagir com ele por meio de atuadores. Um agente é dito inteligente quando realiza a melhor ação possível numa dada situação (Russell and Norvig, 2009).

O ramo da IA possui diversas abordagens ao tratar do tema dos agentes inteligentes. O presente trabalho trata de duas delas, a abordagem dedutiva, na qual o agente realiza suas tarefas orientado ao conhecimento que possui, concretizado por um conjunto de factos que podem derivar novos, a partir do uso de regras bem definidas, e a abordagem indutiva na qual o agente recorre a uma extrapolação cujo modelo pode não ser verificado para alguns dos dados. A abordagem dedutiva relaciona-se com a representação do conhecimento e inferência, oportunidade na qual serão discutidos linguagens de representação do conhecimento como a Lógica Proposicional, *First-Order Logic* (FOL) e *Description Logics* (DL). Os agentes baseados em conhecimento situam-se nesta abordagem. A abordagem indutiva é representada pelo ramo da Aprendizagem Máquina, na qual estão os agentes de aprendizagem. Os agentes reflexivos são dos mais simples de serem implementados e respondem às alterações do ambiente sob a forma de condições.

Um agente reflexivo baseado em modelo possui um estado interno no qual ele tenta manter-se a par do que acontece no ambiente em que atua. Este estado interno reflete as alterações que se verificaram no ambiente e é utilizado para representar a compreensão que o agente tem sobre o mundo. Esta compreensão do mundo é o que se chama de modelo.

Como se vê na figura 2.5, o agente obtém informação acerca do ambiente pelos sensores,

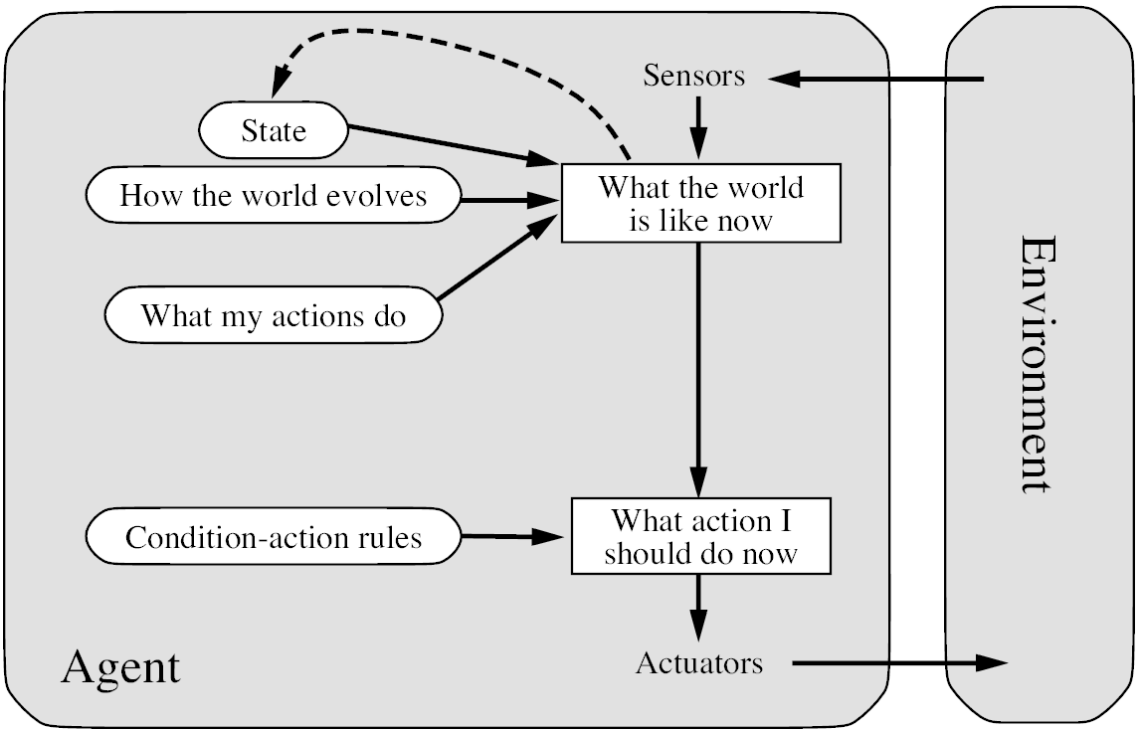



Figura 2.5: *Estrutura de um agente reflexivo baseado em modelo (Russell and Norvig, 2009)*

atualiza o seu modelo do mundo, de modo a perceber como este se alterou e como as próprias ações tiveram parte nesta mudança. Em seguida escolhe dentro de um conjunto de regras de ações condicionadas qual destas deve ser executada com base neste novo conhecimento. Um sistema de automação que guarde um histórico das modificações do ambiente pode ser um exemplo deste tipo de agente.

Agentes baseados em conhecimento são aqueles cujas ações dependem de uma *Knowledge Base* (KB), repositório que possui um conjunto de sentenças ou axiomas que dão validade ao modelo representado pelo sistema. Estes agentes são capazes de adicionar novas informações ao repositório, tipicamente atualizando o estado do modelo, por meio de uma linguagem de representação do conhecimento. O agente consulta então a informação armazenada na KB e consegue extrair informação útil após a realização da atividade de inferência, cuja finalidade é derivar novas sentenças a partir de antigas. A forma como esta atividade é realizada, bem como o significado dado depende das regras utilizadas pelo modelo. Estas regras relacionam-se com a sintaxe e semântica das linguagens de representação de conhecimento disponíveis, existindo diversas destas, que diferem em termos de representatividade dos modelos, avaliação das sentenças ou operadores disponíveis.

```
function KB-AGENT( percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE( percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Figura 2.6: *Exemplo de um agente baseado em conhecimento (Russell and Norvig, 2009)*

Um agente de aprendizagem é caracterizado por quatro componentes (2.7):

- componente de desempenho é responsável por receber informações do ambiente e decidir a ação a ser desempenhada
- componente de aprendizagem é o responsável por melhorar o desempenho do agente
- componente de crítica que avalia o desempenho do agente, contra um padrão determinado e indica o que deve ser feito para ser melhorado
- gerador de problemas que permite ao agente explorar novas situações para melhorar o desempenho no longo prazo

Os três tipos de agentes mencionados acima, a saber os agentes reflexivo, baseado em conhecimento e baseado em aprendizagem, são utilizados neste trabalho para a implementação do sistema pretendido.

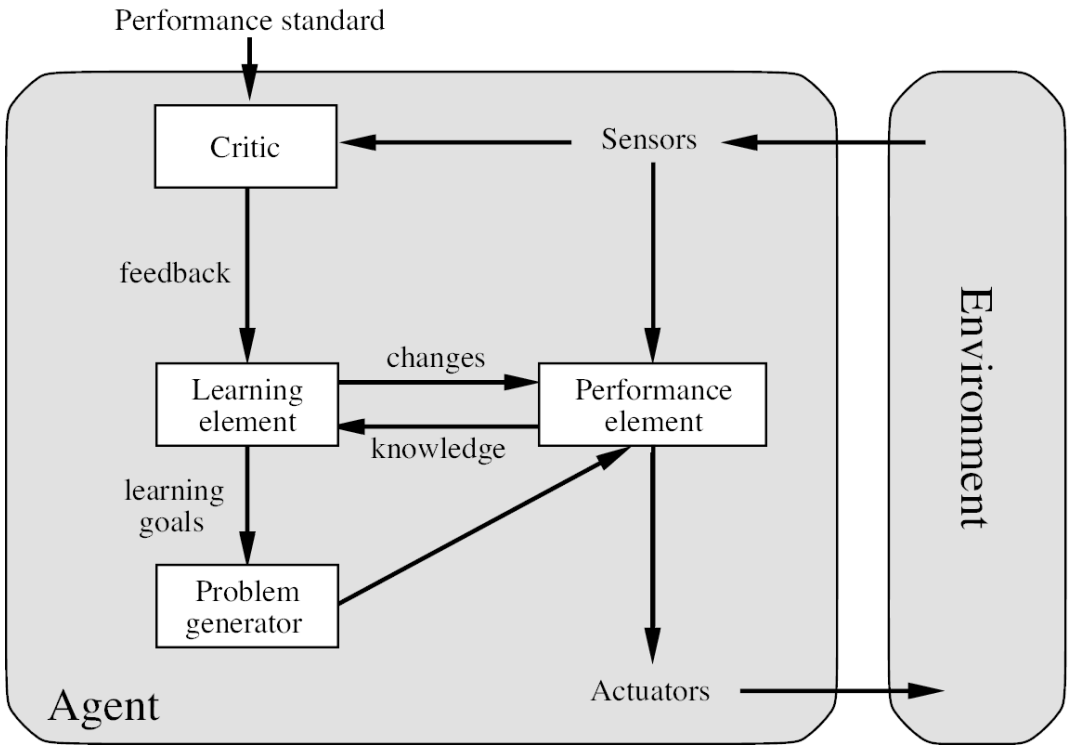



Figura 2.7: *Estrutura de um agente de aprendizagem (Russell and Norvig, 2009)*

### 2.4.2 Aprendizagem Máquina

A Aprendizagem Máquina pode ser definida como a tarefa de fazer com que os computadores sejam capazes de aprender sem serem explicitamente programados para tal. Uma das obras mais antigas sobre o assunto é (Samuel, 1959), na qual o autor postula que "programar computadores para aprenderem com a experiência deve, eventualmente, eliminar a necessidade para esforços detalhados de programação". Em (Russell and Norvig, 2009) encontra-se a definição de um agente de aprendizagem como aquele que "melhora o seu desempenho em tarefas futuras após fazer observações acerca do mundo".

A aprendizagem ocorre de modo indutivo, ou seja, alimenta-se o modelo de mundo do agente de aprendizagem com informações, seus inputs, e obtém-se um resultado, o seu output. Os inputs são comumente vetores de atributos com valores determinados e o agente processa estes dados com recurso a algum algoritmo, quase sempre com acentuado cariz estatístico.

O agente pode ou não ter um retorno acerca dos seus resultados, derivando assim ao menos três classes de algoritmos. Os chamados algoritmos de aprendizagem não supervisionadas, são aqueles em que o agente não recebe qualquer informação acerca do seu atributo output ou classe. Um dos objetivos destes algoritmos é encontrar algum padrão não evidente no conjunto de dados fornecido. A aprendizagem por reforço é aquela no qual o agente aprende a partir de uma série de "estímulos", prêmios ou punições. São

muito utilizados no âmbito dos jogos. Existem, por fim, os algoritmos supervisionados, no qual o agente aprende uma espécie de função que mapeia os inputs em um dado resultado. Estes algoritmos requerem uma fase inicial no qual o modelo é treinado com dados pré-determinados de entrada e saída.
É de notar que os resultados destes algoritmos estão no domínio das probabilidades. No caso específico dos algoritmos de classificação, excetuando-se os casos de sobre-ajuste, há situações em que determinadas instâncias podem resultar em falsos positivos ou negativos. Um dado contexto pode fazer com que um destes resultados ambíguos deva ser evitado sempre que possível, visto que o impacto deste erro pode ser bastante danoso. Exemplo disto é a classificação de um estado como situação normal ou uma emergência, no qual é preferível que haja um falso positivo, ou seja, classificar como uma emergência algo que é uma situação normal, do que o seu contrário. Para tais contextos podem ser utilizadas as matrizes de custo para fazer com que o classificador seja desestimulado a classificar um ambíguo em favor de um valor dito verdadeiro.
Uma das vantagens da Aprendizagem Máquina é lidar com grandes quantidades de dados, o que a torna bastante compatível com o contexto de IoT e *Big Data*. Esta vantagem também é um defeito, visto que, apenas com grandes quantidades de dados disponíveis é possível retirar informação relevante.

### 2.4.3 Representação de conhecimento

Como foi visto na secção anterior, as técnicas de IA baseadas em Aprendizagem Máquina são adequadas em contextos que necessitem de lidar com grandes quantidades de dados. São aptas, também, para realizar exploração deste dados, especialmente a descoberta de padrões não evidentes à partida, ou seja, situações em que o utilizador não possui o controle do ambiente. Um paralelo que se pode fazer, com bastante parcimónia na utilização, é que a utilização de Aprendizagem Máquina é uma espécie de força bruta de resolução de problemas, no sentido em que o modelo é construído após a análise de todo o conjunto de dados disponível. Pode haver alguns contextos que, pela sua natureza, os dados são escassos ou que a uma abordagem exaustiva seja dispensável ou não recomendável. Uma das alternativas à Aprendizagem Máquina é a representação do conhecimento.
Segundo (Poole and Mackworth, 2010), conhecimento é a informação sobre um dado domínio que pode ser utilizada para resolver problemas no referido domínio. A representação do conhecimento é a forma como o conhecimento é utilizado em um agente. Uma *Knowledge Base* (KB) é a representação de todo o conhecimento de um agente. O conhecimento para ser representado necessita de uma sintaxe a ser seguida e uma semântica que atribua significado às entidades que se buscam representar. A sintaxe é definida por linguagens de representação do conhecimento e a semântica é determinada por ontologias. Uma das distinções principais entre a abordagem baseada em conhecimento e outras do

ramo da IA é a utilização de linguagens declarativas, em contraste com as procedimentais. Uma outra distinção, especialmente em relação à Aprendizagem Máquina, é não necessitar de grandes quantidades de dados para gerar as respostas desejadas. A linguagem de representação do conhecimento mais simples é chamada de Lógica Proposicional, que possui como características principais ser declarativa, ou seja, sentenças correspondem aos factos que se quer que sejam representados e ser decidível, uma fórmula pode sempre ser avaliada como verdadeira ou falsa. Uma fórmula existe sempre que há um ou mais factos relacionados por operadores lógicos. Os operadores lógicos da Lógica Proposicional são a negação, disjunção e conjunção.

Uma fórmula é dita válida na Lógica Proposicional se ela for verdadeira para todas as suas interpretações possíveis. Uma fórmula válida é também chamada de tautologia. Um recurso muito utilizado para esta avaliação é a tabela de verdade, em que se lista todas as combinações de possibilidades de valores para todos os átomos de uma fórmula e verifica-se se a tautologia ocorre.

Como já referido, a Lógica Proposicional é a mais simples das linguagens de representação do conhecimento e, portanto, possui uma baixa capacidade expressiva. Uma outra linguagem conhecida como *First-Order Logic* (FOL) busca resolver esta limitação ampliando a quantidade de assunções que um modelo pode possuir. Estas assunções são chamadas de compromissos ontológicos (Russell and Norvig, 2009). Enquanto que na Lógica Proposicional um modelo é apenas formado por factos, na FOL um modelo é formado por factos, objetos e relações. Este compromisso ontológico aliado à introdução dos operadores quantificadores (universais e existenciais) faz com que a FOL seja capaz de criar representações mais concisas e com maior capacidade expressiva. Parte da capacidade expressiva da FOL deve-se à proximidade sintática com as linguagens ditas naturais.

A capacidade expressiva da FOL custa-lhe em termos de decidibilidade, visto que a utilização de alguns operadores lógicos como a disjunçao e a negação fazem com que a inferência sobre factos seja impossível ou de custo elevado. Uma alternativa, então, está na utilização da *Description Logics* (DL), linguagem de representação do conhecimento sucessora das redes semânticas e cujo objetivo foi formalizar semânticas baseadas em lógica (van Harmelen, 2015).

De tudo quanto exposto, resta abordar o tema da ontologia. A palavra tem origem grega, adotada pela filosofia para definir a área que estuda a natureza das coisas, é utilizada em ciência da computação para referir aos vocabulários utilizados por um modelo para representar o mundo. Segundo (Poole and Mackworth, 2010), uma ontologia é uma especificação de significado dos símbolos usados em um sistema de informação, de modo a mapear o estado interno deste sistema e a realidade que se pretende representar. Uma ontologia precede os dados que se obtêm da realidade e define um grau de abstração que tenha significado para as pessoas que a utilizem. Uma ontologia é dotada de uma taxo-

nomia com a existência de conceitos, papéis e indivíduos. Conceitos são conjuntos de indivíduos e os papéis são as relações que conceitos têm entre si.
Uma linguagem de ontologia implementa uma ontologia com suas sintaxes e semânticas próprias. Existem diversas linguagens de ontologia e podem distinguir-se pela linguagem de representação do conhecimento utilizada, sendo FOL e DL as mais utilizadas. O artigo (Lace et al., 2012) traz um levantamento de diversas linguagens de ontologia e faz um comparativo entre as mais importantes, a saber, *Knowledge Interchange Format* (KIF), *Ontology Interchange Language* (OIL), *DARPA Agent Markup Language* (DAML9), *Web Ontology Language* (OWL) e *Resource Description Framework* (RDF). O OWL, devido ao contexto da web semântica, bem como por ser um *standard* da *World Wide Web Consortium* (W3C) tem sido bastante utilizado por pesquisadores de IA e AAL. A existência de ferramentas como o Protégé e o Jena, com *Application Programming Interface* (API) suportado em Java contribuiu para a escolha desta linguagem que será mais detalhada em capítulo adequado.
A existência de uma ontologia, por si só, não traz grandes vantagens sem que seja possível extrair de lá informação útil. Contudo, antes disto é preciso garantir que a própria ontologia seja válida. Uma entidade que cumpre estes dois papéis, garantir validade e extrair informação útil de uma ontologia, é o motor de inferência. O artigo (Sattler et al., 2014) explica as principais tarefas desempenhadas pelos motores de inferência. Primeiramente devem verificar a consistência de uma ontologia. Um exemplo de inconsistência é a existência de uma instância que faça parte de classes que sejam disjuntas. A segunda tarefa de um motor de inferência é verificar a satisfatibillidade da ontologia. Uma ontologia é dita insatisfazível se possuir axiomas contraditórios. Por exemplo, dada uma classe A que possua um predicado p e uma classe B que seja subclasse de A e que proíbe o predicado p, esta ontologia será insatisfazível, porque é impossivel conciliar ambas as condições. A terceira tarefa é verificar as subsunções existentes na ontologia, ou seja, inferir as implicações existentes entre classes que se relacionam. Seria o caso de haver uma classe A que possui uma subclasse B e inferir que toda instância de B também é uma instância de A. Assim, um motor de inferência cumpre os dois papéis mencionados anteriormente. Com os testes de consistência e satisfatibilidade, determina a validade de uma ontologia e com o teste de subsunção, torna explícito uma informação implícita numa KB.
Existem duas abordagens para a implementação de motores de inferência. Uma é orientada ao consequente e outra é baseada em *tableaux*. As orientadas ao consequente utilizam regras de dedução para identificar implicações a partir da ontologia e outros axiomas já inferidos. A abordagem baseada em *tableaux* realiza uma decomposição de cada axioma com base em suas restrições, de modo a criar novos axiomas. Para cada sub-axioma são criadas instâncias para verificar se há consistência e satisfatibilidade do axioma original. Existem várias implementações destes algoritmos, sendo o Pellet um deles, o qual foi adotado neste trabalho.

Resta, ainda, tratar do tema das ontologias temporais, que têm importância no âmbito da representação de atividades. O artigo (Allen, 1983) foi a primeira obra de relevo a discutir a representação temporal do conhecimento, na qual estabelece a chamada álgebra temporal de Allen. Este trabalho propõe uma lista de treze possíveis relações temporais entre dois intervalos de tempo, sendo que cada intervalo é constituído por um instante inicial e um final. A representação gráfica de cada uma das 13 relações encontra-se na figura 2.8. O referido artigo influenciou bastante o ramo da representação do conhecimento, bem como da engenharia de ontologias, sendo a Time-OWL[1], um exemplo disto.

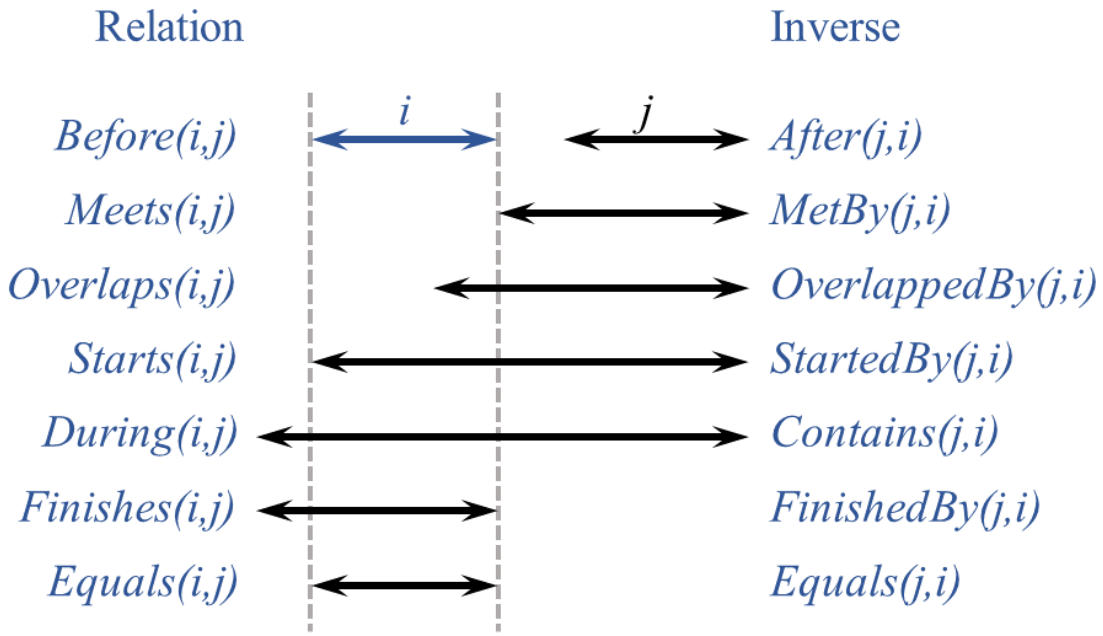



Figura 2.8: *Representação gráfica das treze relações temporais de Allen (W3C)*

**Trabalhos Relacionados:**

Em (Chen et al., 2012), é apresentada uma abordagem para a representação de *Activities of Daily Living* (ADL), atividades de rotina diária, no contexto de casas inteligentes, em tempo real, orientada ao conhecimento. O artigo apresenta três desafios que a representação das ADLs possui: os graus de liberdade dos indivíduos no tocante à ordem de realização das tarefas; a heterogeneidade dos sensores utilizados bem como os formatos dos dados fornecidos; ADLs são compostas por sequências de ações relacionadas temporalmente.

O artigo menciona as diferentes abordagens para a resolução do problema sendo elas: as que utilizam técnicas de análise estatísticas e probabilísticas; as discriminativas, que utilizam técnicas de Aprendizagem Máquina; as que utilizam formalismos lógicos, que

[1] `https://www.w3.org/TR/owl-time/#topology`

utiliza técnicas de Representação do Conhecimento. Descreve, em seguida, o processo de modelagem da ontologia com base no domínio das ADLs, a taxonomia das entidades e a modelagem dos contextos temporais, espaciais e ambientais, cujas informações serão fornecidas pelos sensores.
O artigo (Stavropoulos et al., 2016) apresenta uma *framework* de suporte a pessoas com doenças mentais que utiliza sensores de diferentes modalidades para reconhecimento de atividades diárias.
Acerca de ontologias temporais, (Frasincar et al., 2010) faz uma análise da ontologia Time-OWL, descrevendo as suas três camadas e exemplificando a sua utilização com a representação de uma aquisição com recurso a endividamento (*Leveraged Buyout*).
Em (Batsakis and Petrakis, 2009) é apresentada uma abordagem para representação e inferência no âmbito do OWL 2.0. Distingue a informação temporal em aspecto quantitativo, compreendendo valores para datas, instantes e intervalos, e aspecto qualitativo, considerado informação sobre eventos temporais cujas as durações são desconhecidas, a exemplo das relações temporais de Allen. O aspecto qualitativo é garantido com recurso a regras *Semantic Web Rules Language* (SWRL). O artigo discute então, pormenores acerca do OWL, DL, representação e inferência temporais, utilizando exemplos e algoritmos.

## 2.5 Conclusão

O capítulo traçou linhas gerais acerca de tópicos nos quais o trabalho está enquadrado. Demonstrou a sua adequação ao AAL como o contexto de utilização do sistema, explicitando a relevância desta área de estudo e investigação, seja pelas transformações demográficas que o mundo e, especialmente o continente europeu, virá a experimentar, e, por conseguinte, os investimentos que empresas e organizações nacionais e transnacionais vêm dispendendo para amenizar os efeitos destas transformações.
Tratou-se, ainda, de questões mais afeitas à técnica informática, ao discutir as questões do IoT e aplicações correlatas, bem como os temas da IA, especificamente a conceituação e os tipos de agentes racionais, e os dois grandes métodos de extração de informação relevante a partir de dados, o método dedutivo, na qual a representação do conhecimento se encontra vinculado e o método indutivo, representado pela Aprendizagem Máquina, momento em que se discutiu as respectivas abordagens.
Todos os tópicos tratados estão relacionados, visto que o IoT possibilita a automação de ambientes, bem como a produção de grandes quantidades de dados que podem ser tratados pela Aprendizagem Máquina e, por outro lado, a Representação do Conhecimento possui um papel ao imbuir de significado os conceitos e relações entre as entidades do mundo que se buscam representar. Espera-se, também, que tenha sido demonstrado que as abordagens dedutivas e indutivas possuem suas vantagens e desvantagens e que podem ser manejadas de modo a se complementarem.

# Capítulo 3

# Especificação

## 3.1 Introdução

O sistema proposto pelo presente trabalho tem como objetivos:

- Utilizar condições determinísticas para reagir a situações de baixa complexidade. O escopo deste objetivo é a utilização de automação tradicional.
- Utilizar condições estocásticas para reagir a situação de média complexidade, bem como ser capaz de inferir atividades complexas. O escopo deste objetivo é aliar automação e técnicas de IA, especificamente Aprendizagem Máquina e redes semânticas.
- Realizar análise de comportamento humano com base em padrões de longo prazo. O escopo aqui é utilizar as atividades complexas já inferidas e determinar os hábitos do utilizador do sistema.

Para fins do trabalho, considera-se uma atividade complexa aquela que resulta da composição de atividades simples, por exemplo posturas e outros eventos como a localização numa divisão, a utilização de um dispositivo, num mesmo intervalo de tempo.
O propósito do trabalho, portanto, é definir a arquitetura de um sistema que possua automação ambiental para produzir dados que serão utilizados por motores indutivos (Aprendizagem Máquina) e dedutivos (redes semânticas) para extrair informações úteis, que posteriormente, num grau de abstração mais elevado, serão utilizadas para determinar hábitos do utilizador.
O capítulo descreve a arquitetura do sistema, com a definição dos diferentes módulos que o compõe, bem como a descrição dos respectivos componentes e funções que desempenham. Em seguida elenca os requisitos funcionais, não-funcionais e de sistema necessários para a implementação. Por fim, define os casos de uso que serão utilizados para demonstração das funcionalidades do sistema, na fase de testes e avaliação.

## 3.2 Arquitetura do Sistema

A arquitetura de referência vislumbra, portanto, a existência de três níveis de conhecimento. Na base estão os dados não-processados, oriundos diretamente dos sensores. Acima estão os dados já processados, seja por meio de técnicas de Aprendizagem Máquina, ou por deduções semânticas. No topo da hierarquia estão os padrões de longo prazo que utilizam os referidos dados processados.
A arquitetura do sistema é composta de três módulos, como se vê na figura 3.1. A descrição deles será feita nos tópicos a seguir. São considerados intervenientes do sistema, os cuidadores do utilizador, os profissionais da área de saúde e o próprio utilizador.

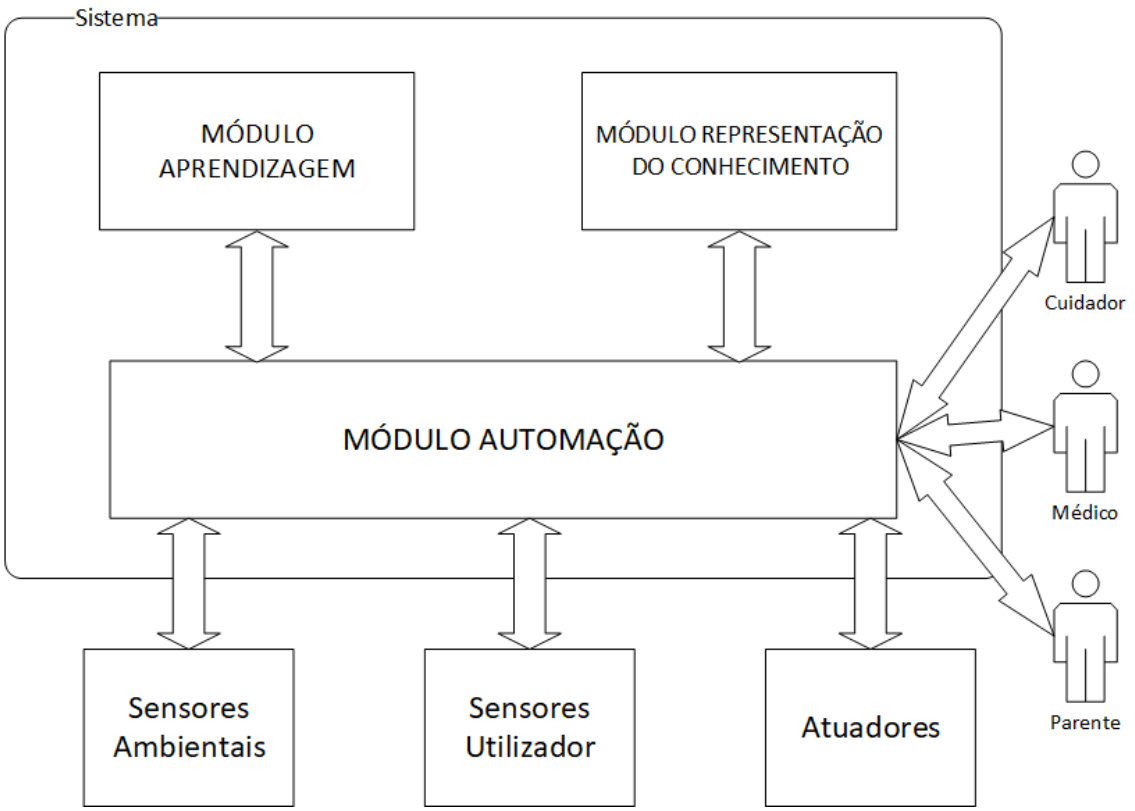



Figura 3.1: *Vista completa da arquitetura do sistema*

### 3.2.1 Arquitetura do Módulo de Automação

O módulo de automação do sistema permite a interação com os dispositivos utilizados para a automação do ambiente, sejam eles sensores ou atuadores, bem como a comunicação com outros módulos e o armazenamento dos dados obtidos. A figura 3.2 detalha os componentes e as interações entre eles. Todos os módulos possuem componentes de comunicação com o exterior, para interagir, seja com os demais módulos, seja com sensores e atuadores, bem como possuem um componente para persistência de dados.

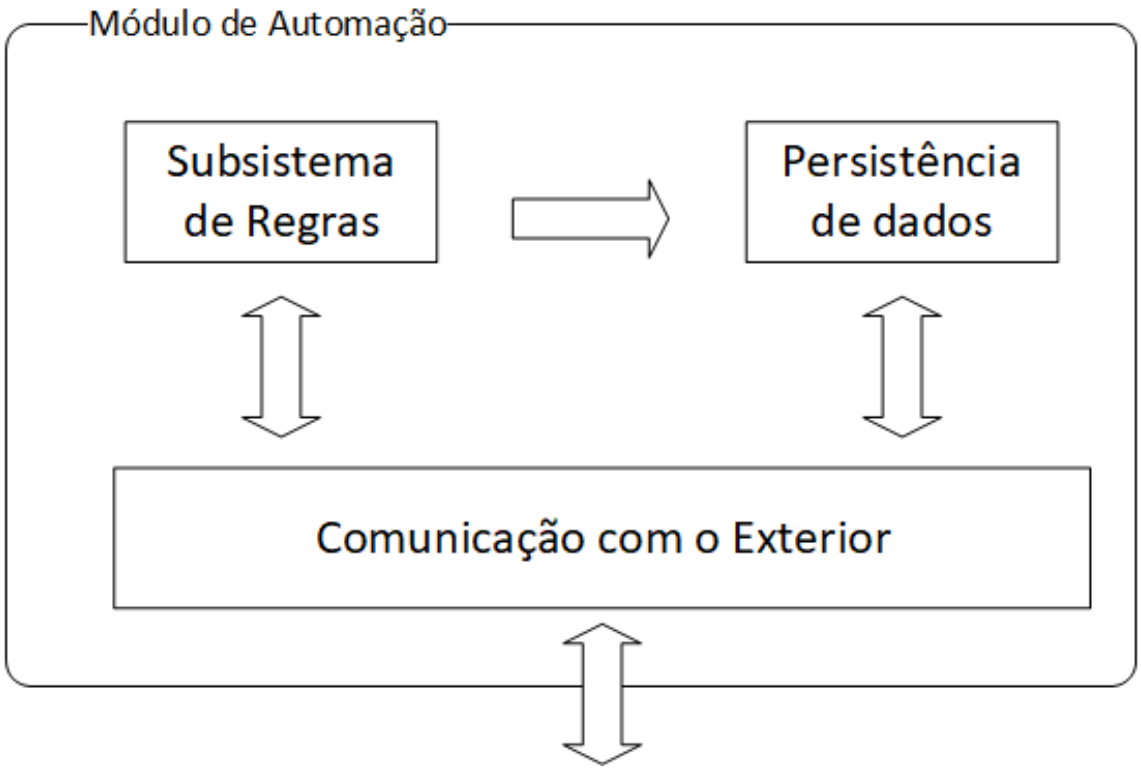



Figura 3.2: *Detalhe da arquitetura do módulo de automação*

**Componentes do Módulo de Automação**

- Comunicação com exterior: compreende qualquer comunicação, seja com os dispositivos, seja com os módulos de aprendizagem e representação do conhecimento, bem como com os intervenientes do sistema. É necessário que o sistema seja capaz de entender diferentes protocolos de comunicação utilizados em IoT e principalmente HTTP para comunicação com os demais módulos.

- Subsistema de regras de automação: este componente realiza a tarefa de comandar os dispositivos conforme regras a serem estipuladas. As condições das regras podem ser determinadas por valores dos sensores, bem como das saídas dos módulos de aprendizagem e representação do conhecimento.

- Persistência de dados: todos os valores dos dispositivos são armazenados em base de dados para garantir a persistência dos mesmos e um histórico que pode ser utilizado para fins de determinação de padrões.

### 3.2.2 Arquitetura do Módulo de Aprendizagem

O módulo de aprendizagem do sistema possibilita a retirada de padrões do ambiente por meio dos dados recebidos do módulo de automação. A figura 3.3 detalha os componentes

e as interações entre eles.

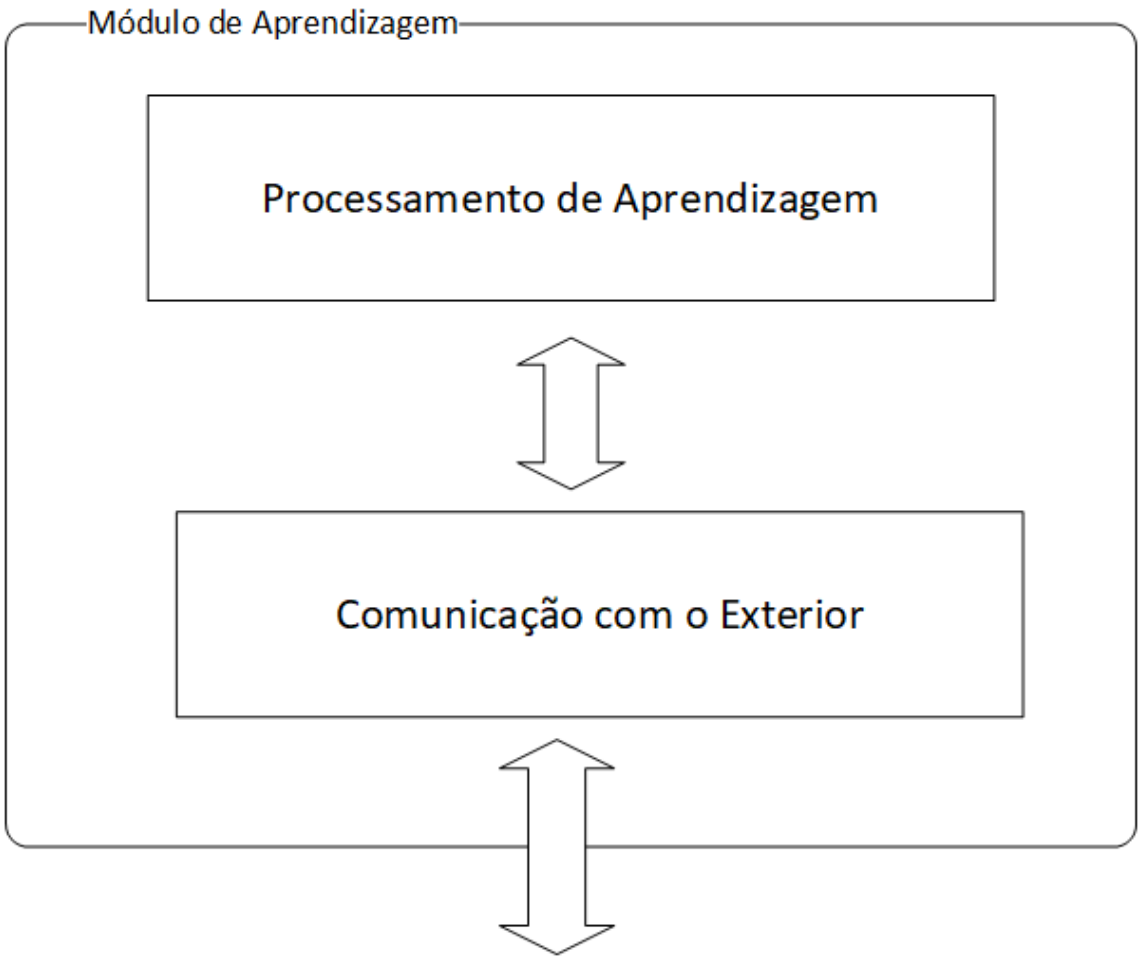



Figura 3.3: *Detalhe da arquitetura do módulo de aprendizagem*

**Componentes do Módulo de Aprendizagem**

- Comunicação com o exterior: este módulo comunica principalmente com os módulos de automação e representação do conhecimento e, eventualmente, com algum outro repositório remoto de dados relevantes. Portanto, basta-lhe que seja capaz de enviar e receber dados por meio de HTTP.
- Processamento de aprendizagem: componente responsável pela transformação de dados não-processados em informação relevante.

### 3.2.3 Arquitetura do Módulo de Representação do Conhecimento

O módulo de representação do conhecimento é o responsável por dar significado de alto nível aos valores obtidos dos sensores, possuindo um modelo de dados que represente a realidade a ser descrita, seja em termos de valores fisiológicos ou ambientais, ou mesmo

de comportamento do utilizador. Além disto, possui suporte a inferências de factos novos a partir de já existentes. A figura 3.4 detalha os componentes e as interações entre eles.

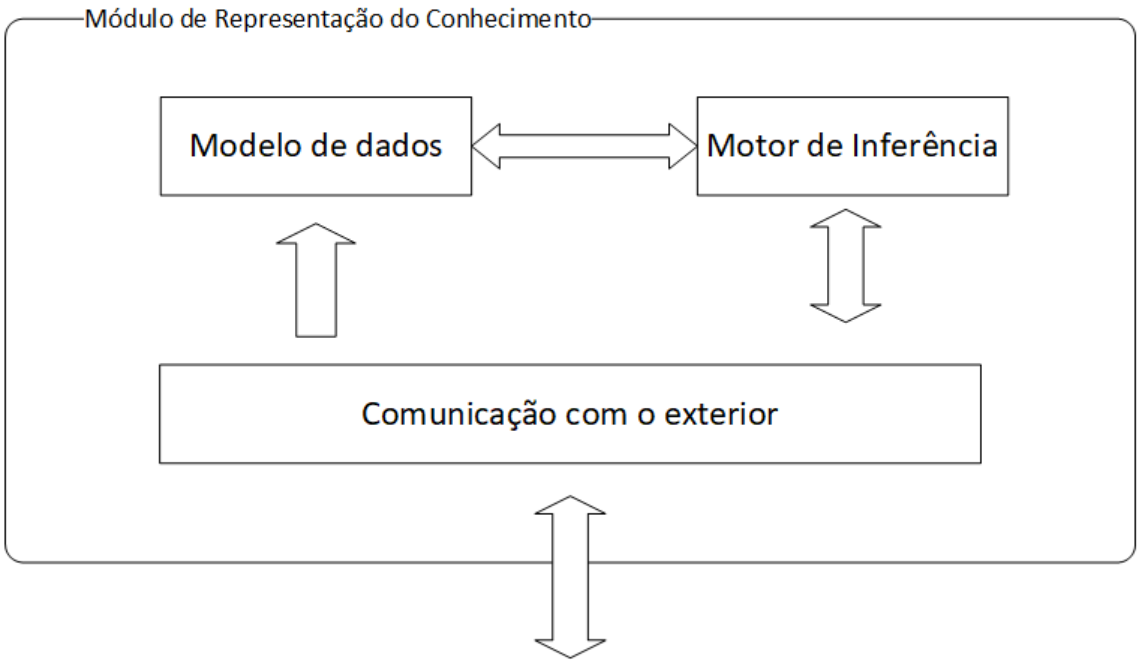



Figura 3.4: *Detalhe da arquitetura do módulo de representação do conhecimento*

**Componentes do Módulo de Representação do Conhecimento**

- Comunicação com o exterior: este módulo irá comunicar principalmente com os módulos de automação e aprendizagem. Portanto, basta-lhe que seja capaz de enviar e receber dados por meio de HTTP.
- Modelo de dados: o módulo deve ser capaz de aceder ao modelo de dados previamente definido de modo a criar novas instâncias de uma dada classe. A criação de novas instâncias pode ser determinada por interações com os demais módulos.
- Motor de Inferência: o módulo deve ser capaz de realizar atividade de inferências com base em regras predeterminadas. A cada criação de uma nova instância deve-se fazer uma consulta à KB para determinar se há alguma nova inferência que possa ser obtida.

## 3.3 Contexto do utilizador

O utilizador vive sozinho e possui alguns sensores individuais, dentre os quais um que identifica sua postura corporal ou atividade simples e outro para monitorar parâmetros cardíacos. A sua casa possui uma gama de sensores para registar dados do ambiente,

especificamente, temperatura, humidade, luminosidade, presença, bem como um sistema central para receber toda a informação, realizar o tratamento, controlar o ambiente e enviar alertas a cuidadores, parentes e médicos.
Além de sensores, o utilizador possui um smartphone para receber notificações e enviar outros dados para o sistema.

## 3.4 Requisitos

### 3.4.1 Requisitos funcionais

**Comuns a todos os módulos**
**ID:** RF1.1
**Título:** Comunicação com demais módulos
**Descrição:** Todos os módulos são capazes de comunicar uns com os outros, de modo que todos possuem um servidor HTTP para o efeito.
**Módulo Automação**
**ID:** RF2.1
**Título:** Comunicação com dispositivos
**Descrição:** O módulo é capaz de interagir com dispositivos existentes na rede, de modo que pode receber informações de sensores e enviar comandos a atuadores. Para isto suporta protocolos de comunicação comuns em IoT como ZWave, Bluetooth e série.
**ID:** RF2.2
**Título:** Automação de tarefas
**Descrição:** O módulo é capaz de reagir, de forma autónoma, a modificações do estado de dispositivos da rede seja para enviar comandos a atuadores ou para enviar dados aos demais módulos do sistema. É capaz, também, de reagir a modificações oriundas dos outros módulos.
**ID:** RF2.3
**Título:** Persistência de dados
**Descrição:** O módulo é capaz de armazenar dados dos dispositivos de forma persistente, com recurso a um *Sistema de Gestão de Bases de Dados* (SGBD).
**Módulo Representação do Conhecimento**
**ID:** RF3.1
**Título:** Manipulação de ontologia
**Descrição:** O módulo é capaz de compreender modelos de ontologias, de modo que pode criar novas instâncias de entidades existentes conforme comando vindo de algum dos outros módulos.
**ID:** RF3.2
**Título:** Dedução de factos

**Descrição:** O módulo é capaz deduzir factos novos a partir dos já existentes. Para esta tarefa possui um motor de inferência que reage a regras lógicas estabelecidas previamente.
**ID:** RF3.3
**Título:** Reatividade a novas deduções
**Descrição:** O módulo possui gatilhos que são ativados quando se verifica a ocorrência de uma nova dedução, permitindo automatizar tarefas baseadas nestas novas deduções.
**Módulo Aprendizagem**
**ID:** RF4.1
**Título:** Algoritmos Aprendizagem Máquina
**Descrição:** O módulo possui implementação de diversos tipos de algoritmos de Aprendizagem Máquina, especialmente os de classificação não supervionada.

### 3.4.2 Requisitos não funcionais

**ID:** RNF1.1
**Título:** Baixo custo
**Descrição:** Todas as aplicações e frameworks utilizadas são gratuitas e os equipamentos necessários são de baixo custo.
**ID:** RNF1.2
**Título:** Escalabilidade
**Descrição:** O sistema permite a inclusão de novos dispositivos e funcionalidades de forma fácil e segura.

### 3.4.3 Requisitos de Sistema (Software e Hardware)

**ID:** RS1.1
**Título:** Dispositivo computacional de baixo consumo
**Descrição:** Foi utilizada uma placa de desenvolvimento Jetson TX1 para a implantação de todos os módulos do sistema.
**ID:** RS1.2
**Título:** Linguagem de programação de alto nível
**Descrição:** As aplicações que suportam as funcionalidades dos módulos do sistema foram implementadas em Java, tomando partido da sua grande popularidade na comunidade open source e pela consequente oferta de soluções gratuitas.
**ID:** RS1.3
**Título:** Aplicações/Tecnologias de comunicação
**Descrição:** O sistema utiliza bastante a comunicação via HTTP na comunicação entre módulos e protocolos IoT, como ZWave e série, para comunicação com sensores e atuadores.
**ID:** RS1.4

**Título:** Base de dados
**Descrição:** A persistência de dados dos sensores e atuadores é realizada com recurso ao SGBD MariaDB.
**ID:** RS1.5
**Título:** Algoritmos de Aprendizagem Máquina
**Descrição:** Os algoritmos Aprendizagem Máquina necessários para o sistema foram implementados pela biblioteca Weka.

## 3.5 Casos de Uso

**ID:** UC1
**Título:** Controle da toma de medicações
**Descrição:** O sistema alerta o utilizador acerca dos momentos de toma de medicações e reporta ao cuidador quando houver uma falha da toma.
**ID:** UC2
**Título:** Controle de parâmetros do ambiente
**Descrição:** Quando o utilizador liga a televisão, o sistema adequa o ambiente a padrões pré determinados de iluminação e temperatura.
**ID:** UC3
**Título:** Acompanhamento dos parâmetros cardíacos
**Descrição:** O sistema identifica qual a atividade desempenhada pelo utilizador e classifica se os seus padrões cardíacos estão em conformidade com os valores habituais e notifica o cuidador em caso de anormalidade.
O sistema utiliza duas estratégias para monitorar o utilizador. Uma baseada em regras determinísticas tratadas exclusivamente pelo módulo de automação. Outra baseada em métodos de Aprendizagem Máquina, na qual o sistema busca aprender os hábitos do utilizador e desencadear ações em casos de desconformidades. Esta estratégia necessita da interação entre os módulos de automação, representação do conhecimento e aprendizagem.

### 3.5.1 Estratégia Baseada em Regras Determinísticas

O primeiro caso de uso proposto para esta estratégia será a utilização de televisão com parâmetros de luminosidade e temperatura determinados pelo utilizador.
O módulo de automação deteta a presença do utilizador na sala, que o utilizador está a ver televisão (simulada pela aplicação **Kodi**), sabe os padrões de ambiente com os sensores de temperatura e luminosidade e pode atuar em aparelhos como ar-condicionado ou aquecedor portátil para ajustar a temperatura ambiente ao valor escolhido pelo utilizador.

O controle destes aparelhos será feito por meio de emissor infra-vermelho ou tomada **Z-Wave**. O sistema controlará a luminosidade por meio de lâmpadas inteligentes em que há o controle da intensidade da luminância.
O segundo caso é a determinação da toma de remédio, por meio de um dispensador automatizado e controlado pelo sistema central. O sistema tem acesso aos horários de toma do remédio, incluídos pelo cuidador no **CalDAV** do utilizador, notifica o utilizador da necessidade de tomar o remédio, seja por meio de ligação ao *smartphone* (utilizando um servidor **Asterix**), ou por mensagem enviada ao **Kodi**, caso este esteja a ver televisão. Além destas notificações, o dispensador possui um sistema de alarme sonoro que é desligado quando o utilizador retira o remédio. Se, após um determinado período de tempo, o alarme não for desligado, o sistema assume que houve uma falha na toma do remédio e envia um alerta ao cuidador.

### 3.5.2 Estratégia Baseada em ML

O primeiro caso de uso proposto para esta estratégia será o de inferir padrões cardíacos anormais do utilizador, consoante a atividade desempenhada.
Para os fins deste trabalho será feita uma distinção entre atividades ditas simples e as complexas. A identificação da atividade simples é realizada por um aparelho, afixado à cintura do utilizador, que é capaz de reconhecer, dentre cinco atividades ou posturas, a saber, estar sentado, em pé, a andar, a correr e deitado. Uma atividade complexa pode ser considerada como a ocorrência, em simultâneo, de um determinado número de eventos, dentre os quais atividades simples. A identificação de uma atividade complexa é feita pelo módulo de representação do conhecimento com base nas informações recebidas pelo módulo de automação.
A classificação é feita em termos de normalidade ou anormalidade da frequência cardíaca do utilizador, conforme cada atividade complexa desempenhada. Assim, o módulo de representação do conhecimento envia ao módulo de aprendizagem as informações relevantes e determina se há ou não anormalidade no ritmo cardíaco do utilizador. Em caso positivo, o módulo de automação realiza as ações de alerta aos intervenientes.
A justificativa para este caso de uso é que a frequência cardíaca varia muito de pessoa para pessoa e conforme cada atividade, de modo que a segmentação da classificação com base neste critério pode ser valiosa em termos de monitoração da saúde de pessoas idosas.
O módulo de automação receberá os dados do sensor de atividades e do sensor de frequências cardíacas e enviá-los-á ao módulo de aprendizagem para avaliar se o comportamento é ou não normal. No último caso, um alerta será enviado ao cuidador. O sistema também irá elaborar relatórios periódicos dos padrões cardíacos, que serão enviados, por email, ao médico.

## 3.6 Conclusão

Neste capítulo foi proposta uma arquitetura de sistema AAL que visa tirar partido da quantidade de dados gerados por dispositivos IoT, geridos por um sistema de automação e integrado a aplicações que implementam técnicas de IA, especificamente, Aprendizagem Máquina e Representação do Conhecimento. Viu-se os três componentes do sistema, as tarefas que cada um desempenha, bem como as dependências necessárias para tal.
Foram elencados os requisitos deste sistema, o enquadramento do utilizador em seu contexto de uso e definidos os casos de uso que refletem os diferentes graus de interação que os módulos do sistema podem possuir de modo a cumprir cada um deste cenários.

# Capítulo 4

# Implementação

## 4.1 Introdução

O presente capítulo trata do modo como os módulos de automação, representação do conhecimento e aprendizagem foram implementados para realizar as tarefas requeridas pelos casos de uso descritos no capítulo anterior.
A secção sobre o módulo de automação apresenta a aplicação *OpenHAB*, suas funcionalidades, algumas linhas acerca da sua configuração para comunicação com sensores e atuadores e exemplos de regras de automação utilizadas.
A secção sobre o módulo de aprendizagem trata da biblioteca de Aprendizagem Máquina WEKA, demonstra a utilização da sua aplicação de interface gráfica e como a sua API foi utilizada para implementar este módulo.
A secção sobre o módulo de representação do conhecimento faz uma exposição sobre linguagens de ontologia, especificamente o RDF, *RDF Schema* (RDFS), OWL e a linguagem de regras SWRL. São apresentadas a aplicação Protégé, para modelagem de ontologias, e a API Jena, para a manipulação destas ontologias, com inclusão de instâncias e realização de inferências.

## 4.2 Módulo de Automação

A implementação deste módulo foi feita com recurso à framework de automação de ambientes OpenHAB[1]. OpenHAB é um software Open Source que centraliza as comunicações com os mais variados tipos de tecnologias e protocolos comummente utilizados para automação e IoT, permitindo, ainda, a gestão das funcionalidades dos dispositivos e dos dados por eles gerados. A maior vantagem desta aplicação é reduzir o esforço ao lidar com estes protocolos. Por ter sido desenvolvida com a framework *Open Services Gateway*

[1] `https://www.OpenHAB.org/`

*Initiative* (OSGi) Equinox[1] possui uma arquitetura modularizada (Figura 4.1), de modo a permitir que alguns componentes possam ser iniciados ou interrompidos em tempo de execução sem comprometer a aplicação.

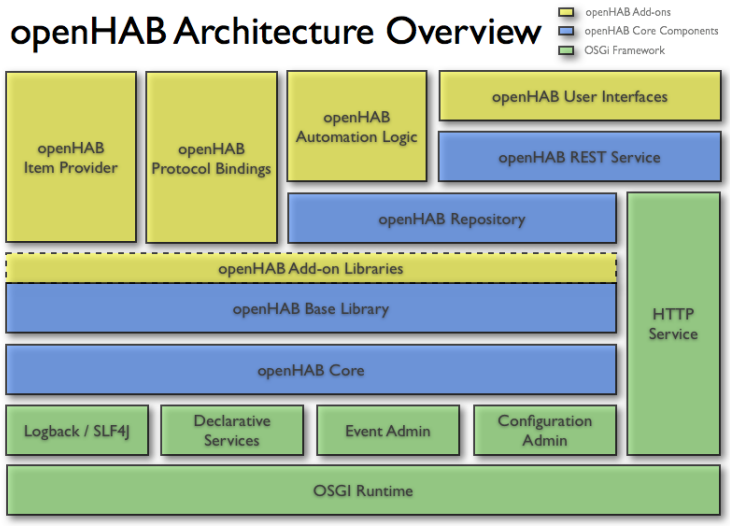



Figura 4.1: *Diagrama de blocos do OpenHAB(OpenHAB, b)*

Como o OpenHAB já possui suporte a diversos tipos de persistência, com especial destaque para bases de dados, além de possuir um servidor web (Jetty) como dependência, os componentes de comunicação (RF1.1) e de persistência (RF2.3) deste módulo ficam desde já resolvidos.

A estratégia utilizada pelo OpenHAB para suportar diferentes protocolos de comunicação é a de abstrair valores e ações destes dispositivos por meio de uma entidade chamada *Item*. Desta maneira um *Item* contém apenas o estado da funcionalidade, que é um valor dentro de um domínio determinado pelo tipo do *Item*, e uma marca temporal. É importante ressaltar que um dado dispositivo pode dar origem a múltiplos *Items*. O estado de um *Item* pode ser obtido por uma funcionalidade de um dispositivo físico (sensores, atuadores), por aplicações (Kodi, Asterisk, etc.) ou por outras fontes de dados (p. ex. uma API).

O OpenHAB comunica com os dispositivos por meio de componentes que sabem realizar a tradução do protocolo utilizado para um dado *Item*. Estes componentes são chamados de *Bindings*. Exemplos de *Bindings* são os que permitem suporte ao protocolo ZWave, Bluetooth, Serial, NTP e outros.

A tarefa de automação é realizado por um motor cuja lógica é realizada pelas *Rules*, scripts com sintaxe Xtend[2], em que se declara uma condição para a execução de ações que podem ser enviar comandos a um *Item* ou mesmo enviar informação ao exterior via HTTP, por exemplo.

[1] `http://www.eclipse.org/equinox/`
[2] `http://www.eclipse.org/xtend/`

As principais funcionalidades deste módulo, portanto, são:

- receber os dados concretos provenientes dos sensores e distribuí-los aos diferentes módulos;
- receber comandos dos diferentes módulos e enviá-los aos atuadores correspondentes.

O objetivo do módulo é transformar o OpenHAB num intermediário entre a base da pilha arquitetural do sistema e as camadas superiores que realizam os processamentos dos dados, seja por indução ou por dedução.

### 4.2.1 Parametrização

A tarefa de parametrização do módulo compreende duas atividades distintas. A primeira consiste em realizar a configuração do OpenHAB de modo a possibilitar a leitura de dados originários de sensores e aplicações, bem como o envio de comandos a atuadores e aplicações. A segunda tarefa relaciona-se com o motor de automação que deve conter a lógica para o envio de dados aos outros módulos ou comandos aos dispositivos, caso uma condição determinada seja satisfeita.

#### Configuração

A configuração do OpenHAB tem por objetivo habilitar os *Bindings*, correspondentes aos protocolos de comunicação ou aplicações, de modo a permitir interação entre a framework e dispositivos ou softwares. A entidade mais importante neste contexto é o *Item* que representa uma funcionalidade específica e que consiste numa marca temporal associada a um valor determinado desta funcionalidade. Ao estabelecer a comunicação do OpenHAB com uma "coisa" e a configuração de um *Item* relativamente a uma funcionalidade desta "coisa", este *Item* estará sempre a ser atualizado de modo a refletir o estado atual da funcionalidade.
Para a tarefa de configuração são utilizados ficheiros de texto localizados em diretórios específicos da aplicação. O ficheiro principal para a configuração é o OpenHAB.cfg encontrado na raiz do diretório de configurações. A sintaxe do ficheiro respeita a lógica *<namespace>:<parametro>=<valor>*, na qual o namespace pode referir a uma funcionalidade nativa do OpenHAB ou derivada de um *Binding*. A configuração de um *Binding* é basicamente indicar ao OpenHAB como comunicar com o dispositivo ou aplicação. Por exemplo, para habilitar a comunicação com dispositivos ZWave (figura 4.2a) é necessário indicar ao OpenHAB a localização do controlador que, por ser um dispositivo USB, deverá ser uma porta série. Para a configuração do Kodi (figura 4.2b) deverá ser informado o endereço IP da máquina, bem como a porta TCP e as credenciais de acesso atribuídas

pela aplicação. Cabe ressaltar que *jetsonKodi* é um identificador a ser referido nos *Items*, visto que é possível ter múltiplas instâncias da aplicação numa mesma rede.

```
zwave:port=/dev/ttyUSB0
zwave:setSUC=true
zwave:masterController=true
```

(a) *Configuração ZWave*

```
xbmc:jetsonKodi.host=10.11.254.200
xbmc:jetsonKodi.wsPort=8086
xbmc:jetsonKodi.username=kodi
xbmc:jetsonKodi.password=kodi
```

(b) *Configuração Kodi*

Figura 4.2: *Exemplos de configurações OpenHAB*

Em sendo uma funcionalidade não nativa do OpenHAB é necessário, ainda, incluir o ficheiro .jar que implementa o *Binding* no diretório de addons para permitir a comunicação. A lista de *Bindings* oficiais é extensa e pode ser encontrada na página wiki[1] do projeto.
A configuração dos *Items* também é feita por meio de ficheiros de texto que devem estar localizados na pasta adequada, com terminação *.items* , podendo possuir múltiplos items. Cada item deve sempre possuir pelo menos o tipo e o nome. O tipo de um item determina o domínio de valores suportados, podendo ser booleanos, *strings* ou números. O nome de um *Item* é utilizado para referí-lo em outros contextos do OpenHAB, especialmente nas *Rules*.
Para referir uma funcionalidade específica é necessário configurar o *Item* em confomidade com o *Binding* utilizado. A diversidade de protocolos praticamente impossibilita uma uniformidade dos parâmetros dos *Items*. Por exemplo, o *Binding* ZWave faz com que o OpenHAB consiga comunicar com o controlador da rede, porém esta pode possuir diversos sensores ligados. Um sensor, por sua vez, pode possuir diversas funcionalidades. Assim a configuração de um *Item* para uma funcionalidade ZWave (figura 4.3a) necessita referir o Id do sensor, bem como a sua funcionalidade. Para a configuração de *Items* do Kodi (figura 4.3b) deve informar-se o identificador da aplicação, no caso *jetsonKodi*, bem como a funcionalidade que se deseja aceder. A configuração de um *Item* que comunique via série (figura 4.3c) necessita apenas mencionar qual a porta do dispositivo.
Todos os *Items* devem ser configurados em conformidade com a sintaxe dada pelo *Addon* que sabe comunicar com o dispositivo ou aplicação fonte dos dados a que se deseja enviar um comando. Uma vantagem que o OpenHAB possui, todavia, é poder interagir com os *Items* por meio de pedidos HTTP, fazendo uso do servidor web presente na ferramenta. Esta funcionalidade pode ser utilizada como uma forma de guardar informações oriundas dos outros módulos do sistema, de modo a possibilitar a sincronização de todos eles. Assim, podem ser criados alguns *Items*, propositadamente sem configuração, apenas para cumprir este papel de sincronizar o módulo de automação com os demais.
O facto de ter uma comunidade bastante ativa, bem como a existência de documentação adequada, wikis e fóruns de discussão foram bastante importantes nesta etapa do trabalho.

[1] `https://github.com/OpenHAB/OpenHAB1-addons/wiki`

```
Number Aeon_Temperature "Temperature [%.1f °C]" (Weather_Chart)
    {zwave="14:command=sensor_multilevel,sensor_type=1"}
Number Aeon_Humidity "Humidity [%d %%]" (Weather_Chart)
    {zwave="14:command=sensor_multilevel,sensor_type=5"}
Number Aeon_Luminance "Luminance [%.0f Lux]"
    {zwave="14:command=sensor_multilevel,sensor_type=3"}
```

(a)

```
String KodiTitle "Title [%s]" {xbmc="<[#jetsonKodi|Player.Title]"}
String KodiState "State [%s]" {xbmc="<[#jetsonKodi|Player.State]"}
String KodiType "Type [%s]" {xbmc="<[#jetsonKodi|Player.Type]"}
String KodiVolume "Volume [%.1f]" {xbmc="<[#jetsonKodi|Application.Volume]"}
String KodiPlayPause {xbmc="<[#jetsonKodi|Player.PlayPause]"}
String KodiNotifiy "Kodi Notifiy [%s]" {xbmc="<[#jetsonKodi|GUI.SHowNotification]"}
```

(b)

```
String Arduino "Arduino [%s]" (arduino) {serial="/dev/ttyACM0"}
```

(c)

Figura 4.3: *Configuração de Items*

### Automação

A automação compreende a especificação de condições para que determinadas ações sejam executadas. A parametrização é feita por meio de entidades denominadas *Rules* que são scripts a serem executados pelo motor de regras do OpenHAB. Estas *Rules* são criadas por ficheiros de extensão *.rules* a serem incluídos no diretório apropriado. Uma *Rule* é constituída por um nome, a sua condição ou *trigger* e o bloco de execução. O nome é arbitrário e apenas interessa do ponto de vista de *Debugging* e *Logging*. A condição pode relacionar-se com *Items*, com regras *Cron* ou por estados do sistema. É permitida a inclusão de diversas condições disjuntas, por exemplo, uma condição que se efetive quando um *Item* mude de estado OU a cada minuto. Assim, é importante frisar que as condições escolhidas podem determinar se a estratégia da automação será periódica ou assíncrona. Sendo uma estratégia periódica, ou seja, ao utilizar-se uma regra *Cron*, o módulo de automação fará *polling* para consultar valores dos sensores, para enviar informação aos outros módulos, ou comandos aos atuadores. Sendo uma estratégia assíncrona, ou seja, ao esperar a modificação de um *Item*, o módulo somente irá realizar aquelas atividades quando o evento condição seja concretizado. Como as condições de estado de sistema são apenas o arranque ou a finalização do sistema, não possuem maior relevância, mas poderiam ser consideradas como membros de uma estratégia assíncrona, cujo evento apenas é realizado uma vez. O trabalho lança mão de ambas as estratégias quando se entende que sejam as mais apropriadas ao contexto de cada caso de uso.
O bloco de execução segue as regras de programação estruturada, sendo permitida a utilização de atribuições, condições e ciclos. A estrutura de um ficheiro *.rules* é formada

por um cabeçalho em que se pode importar classes de bibliotecas, declaração de variáveis globais ao ficheiro e múltiplas regras que podem utilizar estas variáveis e as classes importadas.

As regras são o principal meio de efetivar a comunicação do presente módulo com os demais, especificamente por meio das funções nativas da família sendHttp*Request, que permite realizar os quatro principais pedidos HTTP, a saber, GET, POST, PUT e DELETE. Esta família de funções permite o envio de dados, bem como a escolha do tipo de conteúdo.

Um exemplo ilustrativo desta interação pode ser visto pelo ficheiro *.rules* criado para o envio de dados da banda de ritmos cardíacos para o módulo de aprendizagem. A banda, afixada ao torso do paciente, comunica com uma placa Arduíno que escreve os dados a cada segundo para uma porta USB do servidor em que o OpenHAB está a ser executado. Existe um *Item* configurado para ler os dados escritos pelo Arduíno na porta série.

O objetivo da regra é recolher os dados da banda cardíaca sendo uma amostra por segundo, e, a cada minuto, enviar ao módulo de aprendizagem um conjunto de dados estatísticos destas amostras, a saber, os valores máximo, mínimo, a média e a variância das 60 amostras obtidas. Assim, o ficheiro *.rules* possui algumas variáveis globais para realizar a contagem das amostras, o somatório dos valores e o somatório dos quadrados dos valores. O cabeçalho do ficheiro pode ser visto na figura 4.4.

```
import org.openhab.core.library.type.*
import org.openhab.core.persistence.*
import org.openhab.model.script.actions.*

var float count=0
var float accumulatorQuad=0
var float sum=0
```

Figura 4.4: *Cabeçalho ficheiro .rules*

Existe uma regra chamada "Heart Rate Summations" (4.5), cuja condição é a atualização do *Item* da banda cardíaca, ou seja, a cada vez que o Arduíno escreve algo para a porta série. No bloco de execução faz-se a leitura do valor atual do *Item* e incrementa-se as duas variáveis globais de somatório, além do contador de amostras. Há um filtro passa alto e passa baixo para evitar leituras anómalas.

A segunda regra, chamada de "Heart Rate Stats" (4.6), é executada a cada minuto e com a utilização das variáveis globais obtém-se a média e a variância. Os valores máximos e mínimos são obtidos por uma função nativa do OpenHAB em conjunto com a componente de persistência. Com todos os valores obtidos é criada uma *string* para implementar um *array* em *JavaScript Object Notation* (JSON) que será enviada ao módulo de aprendizado por meio da função *sendHttpPutRequest*, cujos parâmetros são o *Uniform Resource Locator* (URL) do módulo de destino, a definição do tipo de conteúdo como um JSON e o próprio *array*. A regra termina com atribuição de valor 0 a todas as variáveis globais.

```
rule "Heart Rate Summations"

when
    Item HRDetector received update
then
    var float actualValue=(HRDetector.state as DecimalType).floatValue()
    if(actualValue>min_thresh && actualValue<max_thresh){
        count=count+1
        accumulatorQuad=accumulatorQuad+actualValue*actualValue
        sum=sum+actualValue
        stats=stats+actualValue+","
    }
end
```

Figura 4.5: *Regra "Heart Rate Summations"*

Assim, fica demonstrado como o módulo foi implementado e a forma como realiza as suas tarefas.

```
rule "Heart Rate Stats"

when
        System started or Time cron "0 0/1 * 1/1 * ? *"
then
    if(count>1){
        var float max=(HRDetector.maximumSince(now.minusMinutes(1)).state as DecimalType).floatValue
        var float min=(HRDetector.minimumSince(now.minusMinutes(1)).state as DecimalType).floatValue
        var String activity=ActualActivity.state.toString
        var float avg=sum/count

        if(avg > min_thresh && avg < max_thresh){
            var float variance=(accumulatorQuad-(sum*sum/count))/count
            var String array="[\""+activity+"\","+avg+","+min+","+max+","+variance+"]"
            var String result=sendHttpPutRequest("http://localhost:8086/mvapp/","application/json",array)
            if(result.equals("abnormal")){
                logInfo("","Emergencia")
            }
        }
        stats=""
        count=0
        accumulatorQuad=0
        sum=0
    }
end
```

Figura 4.6: *Regra "Heart Rate Stats"*

### 4.2.2 Dispositivos

Para que o módulo de automação seja capaz de realizar as tarefas a que se propõe, é preciso que este comunique com dispositivos que sirvam como fontes de dados. A seguir são elencados, e brevemente descritos, alguns dos dispositivos utilizados no trabalho e os seus propósitos nos casos de uso.

#### Multisensor 4-1

Sensor fabricado pela Aeotec Labs[1] que permite detetar presença de pessoas, bem como a leitura de parâmetros ambientais, especialmente, luminosidade, temperatura e humidade. Este dispositivo pode ser utilizado no caso de uso "Controle de parâmetros do ambiente". A comunicação entre o dispositivo e o módulo de automação é realizada por meio do

[1] https://aeotec.com/

protocolo Z-Wave[1]. Está representado pela figura 4.7.

Figura 4.7: *Multisensor de presença, luminosidade, temperatura e humidade*

**Placa de desenvolvimento Arduíno[2] + Polar *Heart Rate Monitor Interface* (HRMI)[3] + Banda *Eletrocardiograma* (ECG) Polar[4]**

A banda Polar (figura 4.8) faz a leitura dos sinais ECG de um utilizador e comunica com o shield HRMI (figura 4.9) que converte os sinais em valores inteiros que correspondem ao ritmo cardíaco atual. A placa Arduíno recebe estes valores e envia-os ao módulo de automação via porta série. O papel deste conjunto é realizar a leitura de batimentos cardíacos do utilizador para o caso de uso "Acompanhamento dos parâmetros cardíacos".


Figura 4.8: *Banda de leitura de sinais ECG*

[1] http://www.z-wave.com/
[2] https://www.arduino.cc/
[3] https://www.sparkfun.com/products/retired/8661
[4] https://www.polar.com/en

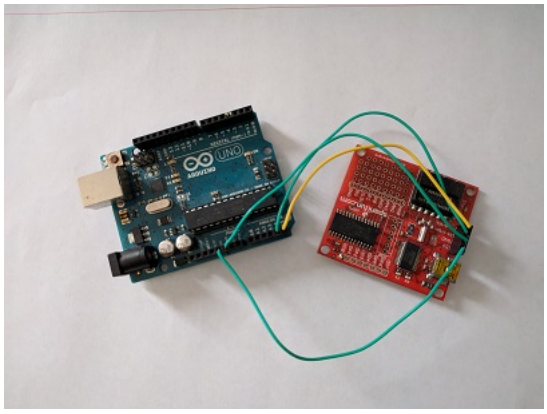


Figura 4.9: *Placa de desenvolvimento Arduíno e Shield Polar HRM*

**Detector de postura**

Dispositivo desenvolvido por (Gonçalves et al., 2009) (figura 4.10) que realiza a detecção da postura do utilizador (estar sentado, em pé, deitado, a andar, a correr, deitado e queda) por meio da análise dos valores de acelerómetros. Comunica com o módulo de automação via porta série.

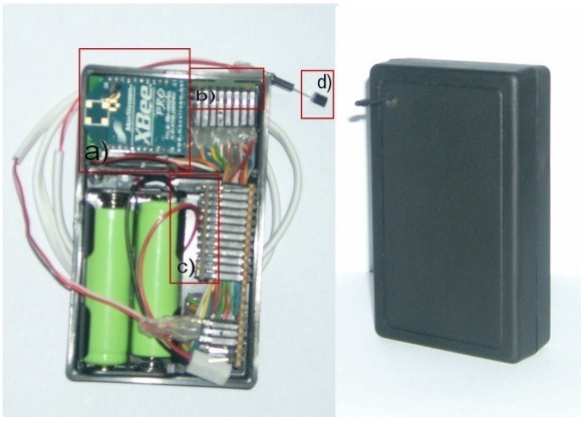



Figura 4.10: *Sensor de postura*

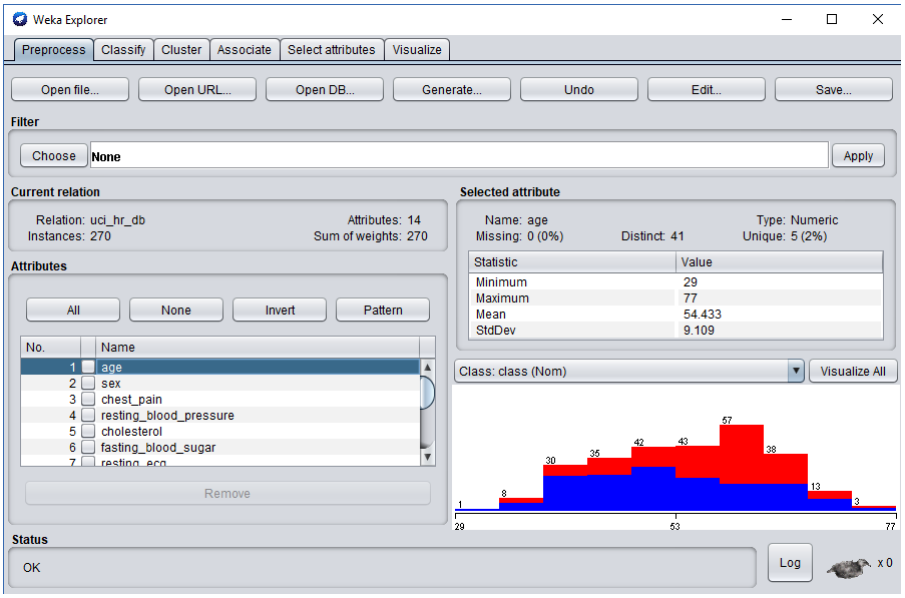
## 4.3 Módulo de Aprendizagem

O módulo de aprendizagem é o responsável do sistema para realizar as tarefas de indução por meio de algoritmos de Aprendizagem Máquina. Para isto ele deverá receber dados vindos do módulo de automação em pedidos HTTP, fazer o processamento destes dados e responder àquele módulo pelo mesmo mecanismo.

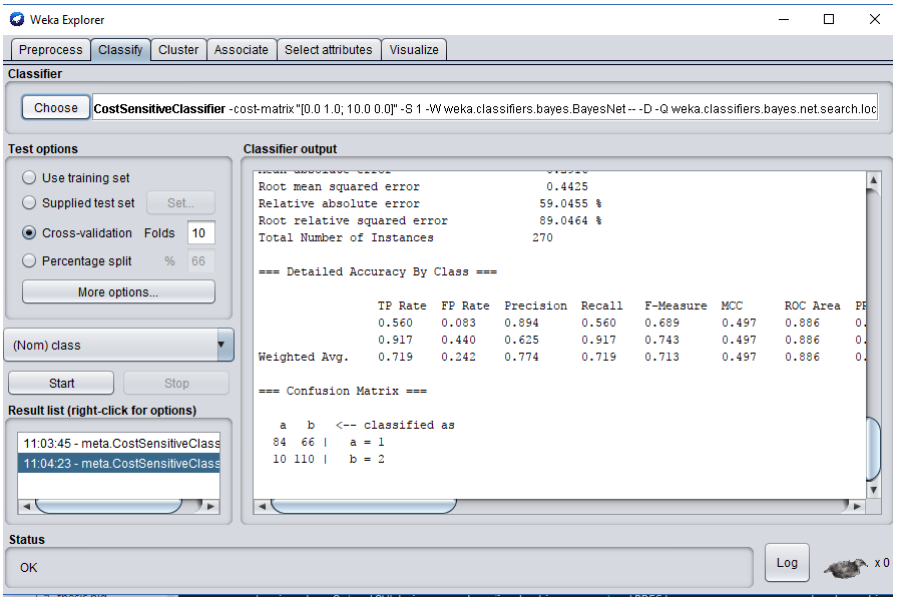
O componente de comunicação com o exterior do módulo foi implementada com o desenvolvimento de uma API *Representational State Transfer* (REST), utilizando o servidor

HTTP Grizzly, cuja principal vantagem é a facilidade de desenvolvimento e implantação visto que o próprio servidor é embebido no ficheiro .jar resultante, sem ser necessário executar um servidor de servlet como o TomCat ou o Glassfish. O processamento dos dados é realizado pela biblioteca de Aprendizagem Máquina WEKA (Frank et al., 2016). O projeto WEKA consiste numa API desenvolvida em Java que permite a utilização de diversos algoritmos de aprendizagem máquina e de uma aplicação com interface gráfica, portanto, mais amigável para o utilizador e útil para o teste do conjunto de dados e escolha dos algoritmos.

As figuras 4.11a e 4.11b são exemplos das funcionalidades disponiblizadas pela interface gráfica Weka, em que, após o fornecimento de um conjunto de dados no formato adequado, apresenta, tanto gráficos sobre a distribuição das classes em termos dos parâmetros existentes, quanto o resultado da classificação com uma diversidade de algoritmos à disposição.




(a) *Visualização das classes em relação aos parâmetros*




(b) *Resultado da classificação do conjunto de dados*

Figura 4.11: *Exemplo da interface gráfica WEKA*

Relativamente à API, o projeto WEKA foi desenvolvido com atenção às boas práticas de desenvolvimento *Orientado a Objetos* (OO), de modo que é facilmente utilizado em outros projetos. A documentação disponibilizada, bem como a comunidade de desenvolvedores também auxíliaram a implementação deste módulo.

O processamento de aprendizagem do módulo é constituído portanto de algumas classes. A classe *ClassifierModel* é constituída de um classificador (*Classifier*), que representa qualquer algoritmo de classificação da biblioteca, um filtro (*Filter*), instâncias de treino (*Instance*) e o estimador (*Evaluation*) que realiza a predição dos novos dados após o

treino. A classe *AbstractModel* é uma composição da classe anterior, mais o conjunto de instâncias adicionadas ao modelo após o treino e um contador destas instâncias. O objetivo desta classe é representar o modelo de classificação para diversos conjuntos de dados que se desejem utilizar. Cada conjunto de dados é composto por parâmetros e classes que podem ser muito diversos entre si. Exemplo disto é o conjunto de dados utilizado no trabalho, que contém como parâmteros, a atividade desempenhada pelo utilizador, a média de seus batimentos cardíacos no período de um minuto, o valor máximo, mínimo e sua variância. A classe deste conjunto de dados admite os valores positivo ou negativo para a normalidade do ritmo cardíaco. Assim, a classe que representa este conjunto de dados estende a classe *AbstractModel* e em seu construtor, além de inicializar o *ClassifierModel* com as informações sobre o classificador e os dados de treino, cria o conjunto de dados de teste, que serão utilizados pelo classificador após o treino, com toda a informação de parâmetros e classes.

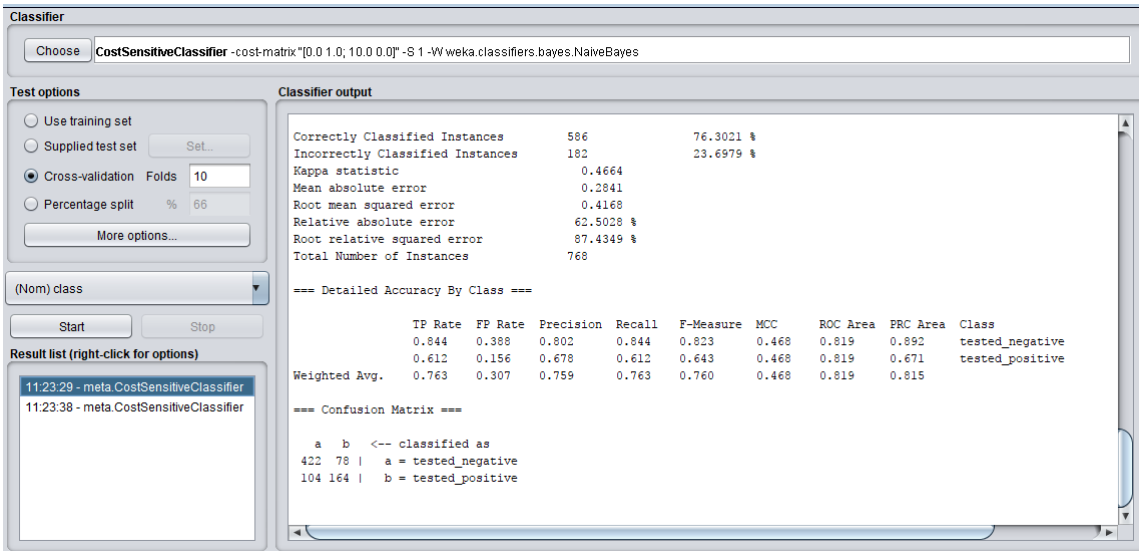
Um *Classifier* pode ser escolhido dentre uma lista fechada de implementações já desenvolvidas pelo projeto Weka. O trabalho utilizou a implementação do *Naive Bayes* e um meta classificador *CostSensitiveClassifier*, que resolverá uma situação de ambiguidade (falsos positivos e negativos), o que é particularmente importante neste contexto visto que um falso positivo significa que o modelo julga que é uma situação de normalidade quando pode ser uma situação de emergência, sendo, pois, preferível que seja lançado um alerta falso em situação de normalidade a não ser lançado um alerta quando pode haver uma anormalidade. Um *CostSensitiveClassifier* aplica uma matriz de custo sobre um classificador específico. Esta matriz de custo possui as dimensões da quantidade de classes do modelo, assim, para duas classes, a matriz tem dimensões 2x2. A diagonal principal da matriz possui os valores positivo verdadeiro (TP) e negativo verdadeiro (TN), enquanto que a diagonal secundária possui os valores ambíguos, falso negativo (FN) e falso positivo (FP), conforme pode ser visto na figura 4.12a. A matriz de custo identidade, aquela que não altera os resultados do classificador consiste na diagonal principal a zeros e a matriz secundária a uns, representada na figura 4.12b. Para o efeito que se pretende, ou seja, favorecer a classificação de um falso positivo como um negativo, é preciso fazer com que o custo do falso positivo seja aumentado, o que pode ser obtido com a matriz exibida na figura 4.12c.

$$\begin{bmatrix} TP & FN \\ FP & TN \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 \\ 10 & 0 \end{bmatrix}$$

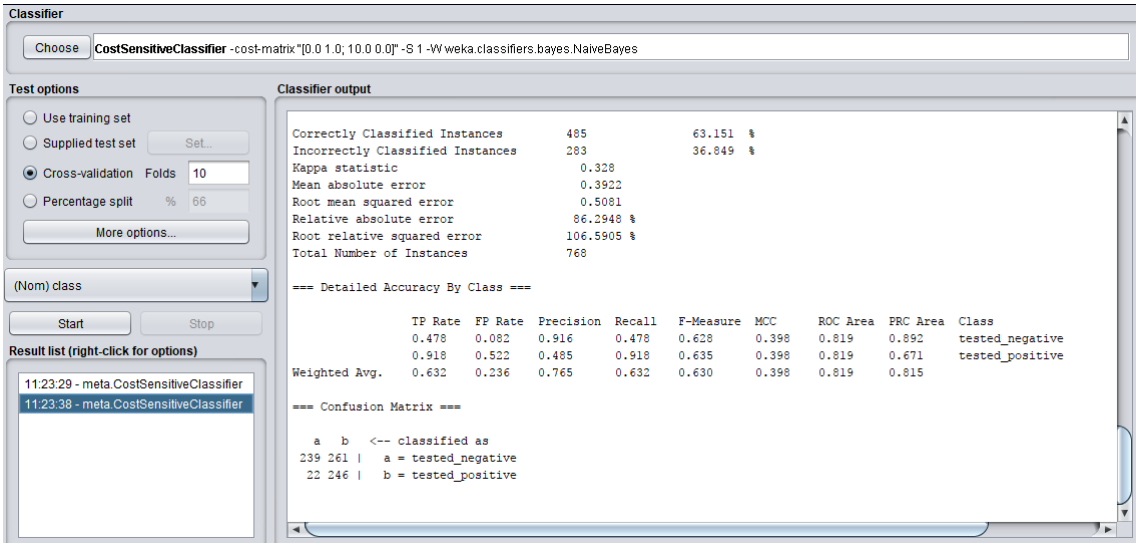
(a) *Matriz custo* (b) *Matriz custo identidade* (c) *Matriz custo utilizada*

Figura 4.12: *Matrizes de custo*

O efeito pode ser verificado nas figuras abaixo, sendo que a 4.13a refere a classificação com uma matriz de custo identidade, onde se vê na matriz de confusão que a quantidade de falsos positivos é igual 104 e de valores negativos verdadeiros é 164, enquanto que figura 4.13b é aplicada a matriz de custo da figura 4.12c e vê-se que os falsos positivos diminuem para 22 e os negativos verdadeiros aumentam para 246.




(a) *Resultado da classificação com matriz de custo identidade*




(b) *Resultado da classificação com matriz de custo 10 vezes maior sobre os falsos positivos*

Figura 4.13: *Resultados de classificações com matrizes de custo*

## 4.4 Módulo de Representação do Conhecimento

O principal objetivo deste módulo é permitir a representação de mais alto nível e com maior valor semântico a partir de dados recebidos do Módulo de Automação. Esta repre-

sentação é feita com recurso a Ontologia, especificamente o OWL, implementado pelas *frameworks* Protégé e Jena. Com a utilização de ontologia OWL é possível criar regras SWRL, com o propósito de inferir atividades complexas a partir da realização de diversas atividades simples em simultâneo. Passa-se, então, à explicação dos tópicos referidos, bem como a descrição da implementação do módulo.

### 4.4.1 Ontologias e OWL

Uma Ontologia pode ser definida como uma descrição formal de um domínio de conceitos, também denominados por classes, que possuem um conjunto de indivíduos ou instâncias e a representação de relacionamentos entre classes ou entre indivíduos. O OWL[1] é uma linguagem de modelação de Ontologia criada originariamente no contexto da Web Semântica[2], pelo W3C, mas que foi adotada por diversas áreas da AI, para fins de representação do conhecimento. O OWL é uma extensão da linguagem RDFS, que por sua vez estende a linguagem RDF e todas são parte da pilha tecnológica da Web Semântica. O SWRL[3] é uma linguagem que combina OWL e *Rule Markup Language* para estender o conjunto de axiomas do OWL de modo a combiná-los com a base de conhecimento da Ontologia. Pode ser utilizado como forma de estipular regras para a inferir a associação de indivíduos a determinadas classes.

O RDF é um modelo de dados baseado em afirmações segundo o padrão Sujeito-Predicado-Objeto, também conhecido como um triplo. Um triplo também pode ser representado como um grafo dirigido, no qual Sujeito e Objeto são nós e o predicado a aresta que os conecta. O predicado (propriedade), é uma relação entre duas entidades (recursos) ou entre um recurso e um literal (valor). Do ponto de vista do RDF, recursos e propriedades são identificados por um *Uniform Resource Identifier* (URI).

O RDF não traz em si quaisquer regras semânticas, cuida apenas de representar afirmações. A atribuição de significado às afirmações é realizada pelo RDFS e pelo OWL, cada qual à sua maneira. Ambas são linguagens de modelação de dados que dependem do RDF para serem implementadas. A existência de regras semânticas permite que algumas afirmações possam ser obtidas a partir de outras pré-existentes, por meio do mecanismo de implicação (em inglês, *entailment*).

O RDFS possibilita a declaração de restrições nos recursos e hierarquização do modelo com Classes, Subclasses e Subpropriedades. As restrições nos recursos podem ser representadas por meio da atribuição de domínios e imagens das propriedades. Assim, considerando o facto de uma Propriedade $p$ possuir a classe $A$ como domínio e a classe $B$ como imagem (em RDFS), a afirmação de que $x\ p\ y$ (em RDF) implica afirmar que $x$ é instância da classe $A$ e $y$ é instância da classe $B$. De igual maneira, o facto de $A$ ser subclasse de $C$ e

[1] `https://www.w3.org/\ac{OWL}/`
[2] `http://www.w3.org/standards/semanticweb/`
[3] `http://www.w3.org/Submission/\ac{SWRL}/`

*x* ser instância de C implica que x seja indivíduo de A. Importante mencionar que o RDFS é uma linguagem fraca o suficiente para permitir que afirmações inconsistentes ou incoerentes sejam feitas, por exemplo, afirmar que um recurso é simultaneamente instância e classe, ou mesmo uma instância pertencer a duas classes que deveriam ser disjuntas.
O OWL traz mais construções semânticas e possibilidade de restrições que o RDFS não consegue lidar. Exemplo disto é a possibilidade de declarar que duas classes são disjuntas entre si, ou seja, que, em sendo A e B disjuntas, uma instância de A nunca pode ser instância de B. Além disto, o OWL permite também a criação de dois tipos de propriedades ou relações, chamadas de *ObjectProperty* e *DataProperty*. Uma relação do primeiro tipo implica que tanto o sujeito quanto o objeto são instâncias. A do segundo tipo implica que o sujeito seja uma instância e o objeto ser um literal, ou seja, um valor compatível com o padrão XML Schema Definition *(XSD)*. É possível, também, definir uma classe a partir de relações de conjuntos, união e interseção, entre outras classes.
Estas e outras regras semânticas definidas pelo modelo de dados do OWL são conhecidas como axiomas. Comummente uma KB em DL é dividida entre o que se chama *TBox*, ou a união entre modelo de dados e axiomas e a *ABox*, as instâncias ou factos RDF. Esta divisão é considerada uma boa prática no ramo da engenharia de ontologias, que cuida da modelização das taxonomias e criação das regras semânticas que serão utilizadas posteriormente para a inferência de novos factos a partir de outros. Inferência (ou *reasoning*) é a atividade possibilitada pelas ontologias que mais interessa à AI e, neste âmbito, especialmente o SWRL.
O OWL em sua versão 1.0 (atualmente existe a especificação do OWL2), possui três sublinguagens: **OWL Full**, **OWL DL** e **OWL Lite**. A diferença entre elas reside no compromisso entre expressividade e decidibilidade. Expressividade entende-se pela possibilidade de criar modelos e relacionamentos complexos. O custo da expressividade é fazer com que o inferência não seja possível, ou seja, o modelo é indecidível. O **OWL Full** utiliza toda a gama de construções permitidas pelo padrão OWL e com isto enfatiza a expressividade do modelo. Em razão disto recomenda-se o seu uso quando não houver interesse pela utilização de inferência. O **OWL DL** restringe a utilização de alguns destas construções de expressividade para permitir que o modelo possa ser decidível por motores de inferência. Um exemplo de restrição é a disjunção entre as *ObjectProperties* e as *DataProperties*. O **OWL Lite** restringe ainda mais a expressividade de modo a que seja um bom ponto de entrada para pessoas inexperientes.

### 4.4.2 SWRL

SWRL é uma linguagem proposta no contexto da Web Semântica de modo a expressar regras e lógica. As regras em SWRL são compostas por duas partes, um antecedente (corpo) e um consequente (cabeça). Quaisquer das partes são compostas por zero ou

mais átomos que podem ser relações unárias ou binárias. SWLR considera como relações unárias as classes e os tipos de dados. São relações binárias as *ObjectProperties* e as *DatatypeProperties*. Os átomos referem-se sempre a instâncias ou variáveis. Um exemplo típico da sintaxe do SWRL é:

$hasParent(?x1, ?x2) \wedge hasBrother(?x2, ?x3) \implies hasUncle(?x1, ?x3)$

Tal lê-se da seguinte forma: caso uma instância x1 possua como objeto da propriedade *hasParent*, uma instância x2, e este possua como objeto da propriedade *hasBrother* a instância x3, então x1 possui como objeto da propriedade *hasUncle*, a instância x3. De modo mais sintético equivale a dizer que se x1 tem um pai, x2 e se x2 possui um irmão, x3, então x3 é tio de x1. Desta maneira, se a KB possuir dois factos, "João" *hasParent* "Maria" e "Maria" *hasBrother* "Paulo", um motor de inferência com a regra acima consegue inferir o facto "João" *hasUncle* "Paulo". Assim, o SWRL representa um ganho de expressividade sem comprometer a decidibilidade do **OWL DL**, pelo que se considera uma ferramenta valiosa para o âmbito da AI.

O objetivo deste trabalho é criar uma série de regras cujas condições permitam a realização de inferências de factos novos a partir dos já existentes, especificamente no que se refere à identificação de atividades complexas.

### 4.4.3 Protégé e Apache Jena

Protégé[1] (Musen and Team, 2015) é um projeto *Open Source* desenvolvido e mantido pela Universidade de Stanford, cujo propósito é auxiliar na tarefa de modelização de Ontologias. Trata-se de uma aplicação com interface gráfica implementada sobre a *framework* OSGi *Equinox*, que utiliza a *OWL API*[2] com possibilidade de criação de *plugins* por terceiros. Suporta ainda utilização de motores de inferência, bem como facilita a criação de regras *SWRL* e consultas *SPARQL*. Possui ainda uma fonte de documentação bastante fiável[3], com diversos tutoriais e descrições das funcionalidades.

Assim, é uma ferramenta valiosa para a criação do modelo de dados, regras e testes sem a sobrecarga de lidar diretamente com a sintaxe XML/RDF. Além disto, por utilizar a *OWL API* permite a manipulação programática do modelo de modo a ser integrável com sistemas automatizados.

Apache Jena[4] é uma *framework* mantida pela fundação Apache[5] para implementação de aplicações para a Web Semântica. O Jena possui API's próprias, em Java, para criação de RDF's, Ontologias OWL, consultas SPARQL e motores de inferência. O que a distingue do Protégé é a integração com bases de dados SDB (em formato SQL) ou TDB (em

[1] http://protege.stanford.edu/
[2] http://owlapi.sourceforge.net/
[3] https://protegewiki.stanford.edu/
[4] https://jena.apache.org/
[5] https://www.apache.org/

formato de triplos) e um servidor HTTP (Fuseki) para servir de *endpoint* para outras aplicações. Uma visão geral da arquitetura é fornecida pela figura 4.14.

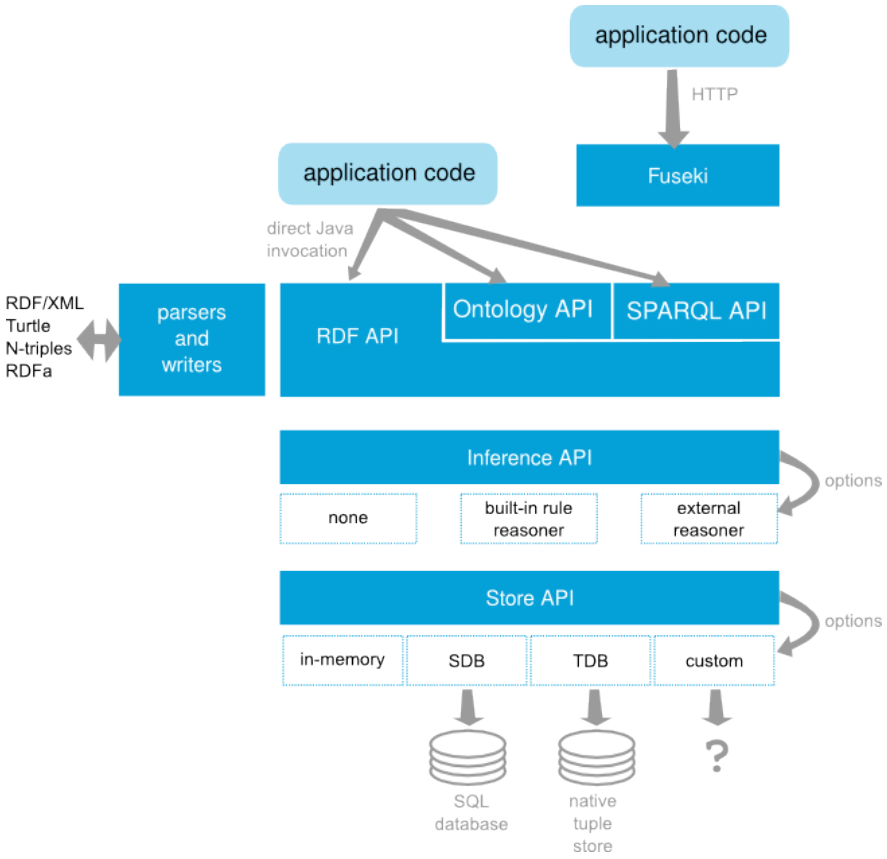



Figura 4.14: *Arquitetura da framework Jena(Apache Foundation)*

As duas ferramentas, apesar de terem propósitos similares, são úteis para duas atividades distintas. Será utilizada a aplicação com interface gráfica do Protégé para a modelização dos dados em classes e relações entre elas, bem como a criação das regras SWRL, de modo a se criar a Ontologia adequada. O Jena será utilizado para permitir a manipulação do modelo criado, ou seja, a criação das instâncias das classes, bem como a realização de inferências com base nas referidas regras SWRL. O motor de inferência a ser utilizado será o Openllet[1] uma implementação do Pellet[2], cuja maior vantagem é permitir a utilização de tipos de dados nas inferências.

### 4.4.4 Modelo de dados da Ontologia

A taxonomia utilizada por este trabalho, criada com recurso à ferramenta Protégé, está representada pela figura 4.15. O modelo possui uma classe raiz chamada de *Event* cujo

[1] https://github.com/Galigator/openllet
[2] http://semanticweb.org/wiki/Pellet

propósito é representar qualquer evento. Um *Event* possui um instante de início (propriedade *hasBeginning*) e um instante de fim (propriedade *hasEnd*). Ambas as propriedades pertencem à Ontologia OWL-Time[1]. *Event* possui quatro subclasses, *Localization*, *Device*, *Activity* e *Emergency*.

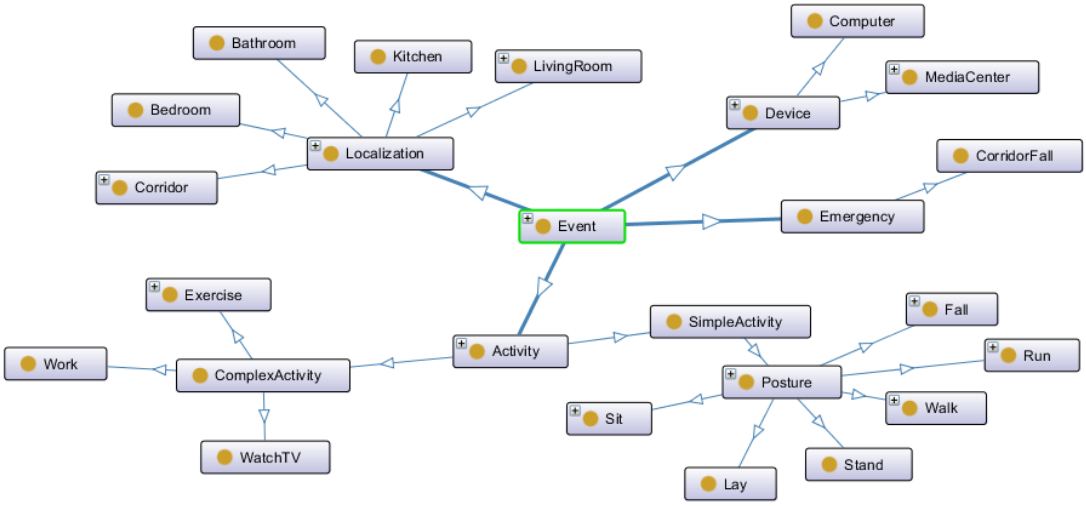



Figura 4.15: *Taxonomia utilizada pelo trabalho*

A classe *Localization* representará a presença numa determinada divisão da casa. Uma das vantagens desta abordagem é "temporalizar" a dimensão espacial. Do ponto de vista do modelo, todo evento está contido na dimensão temporal e qualquer facto ocorrido na dimensão espacial é representado, em termos de sua ocorrência, na dimensão temporal. Assim, é possível representar que a presença de uma pessoa na sala de estar ocorre entre dois instantes temporais, no caso, quando entra e quando sai da referida divisão.
A classe *Device* representa a utilização de um dispositivo qualquer dentro do ambiente. Exemplos de dispositivos relevantes são representados pelas subclasses *TV* e *Computer*.
A classe *Activity* representará as atividades desempenhadas pelo utilizador, sejam elas simples posturas ou mesmo atividades complexas. As informações sobre as posturas do utilizador são dadas pelo sensor que está posicionado na sua cintura, sendo consideradas estar em pé, sentado, a andar, a correr, deitado. O sensor também deteta a queda do utilizador mas, para fins do modelo, tal não é representado como uma postura.
As atividades, bem como as posturas, são consideradas em conjunto como atividades simples ou atómicas, no sentido em que são indivisíveis. O modelo pretende representar também atividades ditas complexas, ou seja, que possam ser compostas por mais do que um evento. Por exemplo, a atividade de assistir à televisão (*WatchTV*) pode ser considerada como a composição de alguns eventos, como o facto do utilizador estar na sala de estar (*LivingRoom*), estar sentado (*Sit*) e ter a televisão ligada (*TV*). A granularidade

[1] `http://www.w3.org/TR/owl-time/`

da descrição de uma atividade complexa pode ser determinada pela existência de mais fontes de dados, ou seja, pode ser incluída a informação de que o utilizador está sentado na poltrona própria para ver televisão.
Independentemente da granularidade ou nível do detalhe que se pretenda conferir a uma atividade complexa, o que importa referir é que uma atividade complexa pode ser representada como eventos contidos uns nos outros, do ponto de vista temporal. Clarificando o ponto, o trabalho parte do pressuposto de que existe mais do que um evento e que em um dado intervalo de tempo eles acontecem em simultâneo. Quando determinados eventos ocorrem em simultâneo, ou quando eles ocorrem durante outros, segundo a álgebra de Allen (Allen, 1983), considera-se a existência de uma atividade complexa. Tomando o exemplo acima, pode-se considerar por ocorrida a atividade *WatchTV* quando o utilizador entra na sala de estar, senta-se e liga a televisão. Do ponto de vista do modelo, esta sequência de eventos é traduzido como, o início de *LivingRoom* ser anterior ao de *Sit* e o deste anterior ao de *TV* e o fim deste ser anterior ao fim de *Sit* e, de igual forma, o fim deste ser anterior ao de *LivingRoom*.
Desta maneira, para fins de identificação de atividades complexas, serão utilizadas regras SWRL. Para o exemplo referido, a regra SWRL correspondente seria:

$$\begin{aligned} &LivingRoom(?x) \wedge Sit(?y) \wedge TV(?z) \wedge hasBeginning(?x,?startX) \wedge hasBeginning(?y,?startY) \wedge \\ &hasBeginning(?z,?startZ) \wedge hasEnd(?x,?endX) \wedge hasEnd(?y,?endY) \wedge hasEnd(?z,?endZ) \wedge \\ &\quad inXSDDateTime(?startX,?time1) \wedge inXSDDateTime(?startY,?time2) \wedge \\ &\quad inXSDDateTime(?startZ,?time3) \wedge inXSDDateTime(?endX,?time6) \wedge \\ &\quad inXSDDateTime(?endY,?time5) \wedge inXSDDateTime(?endZ,?time4) \wedge \\ &swrlb:lessThanOrEqual(?time1,?time2) \wedge swrlb:lessThanOrEqual(?time2,?time3) \wedge \\ &swrlb:lessThanOrEqual(?time3,?time4) \wedge swrlb:lessThanOrEqual(?time4,?time5) \wedge \\ &\quad swrlb:lessThanOrEqual(?time5,?time6) \implies WatchTV(?z) \end{aligned}$$

Os passos intermédios têm relação com o modelo de dados da Ontologia OWL-Time, que considera *hasBeginning* e *hasEnd* como instâncias da classe *Instant* que, por sua vez, possuem como *DatatypeProperty* a relação inXSDDateTime, cuja imagem é o literal xsd:dateTime.
Uma forma de simplificar a lógica subjacente a estas inferências é traduzir as relações temporais de Allen para regras SWRL. Assim, pode-se substituir a regra anterior por uma mais compacta que traduz exatamente o que se pretende:

$$\begin{aligned} &LivingRoom(?x) \wedge Sit(?y) \wedge MediaCenter(?z) \wedge eventDuring(?x,?y) \wedge \\ &\quad eventDuring(?y,?z) \implies WatchTV(?z) \end{aligned}$$

A regra lê-se como, ocorre *WatchTV* quando *Sit* ocorre durante *LivingRoom* e *MediaCenter* ocorre durante *Sit*. A definição da *ObjectProperty eventDuring* é:

$$Event(?x) \wedge Event(?y) \wedge hasBeginning(?x, ?startX) \wedge hasBeginning(?y, ?startY) \wedge$$
$$hasEnd(?x, ?endX) \wedge hasEnd(?y, ?endY) \wedge before(?startX, ?startY) \wedge$$
$$before(?endY, ?endX) \implies eventDuring(?x, ?y)$$

A regra para eventDuring, portanto, ocorre quando, dados dois eventos, o primeiro possui o início anterior/*before* ao início do segundo e o fim do segundo é anterior/*before* ao fim do primeiro. Por fim, a definição da regra para *before*:

$$Instant(?x) \wedge Instant(?y) \wedge inXSDDateTime(?x, ?w) \wedge inXSDDateTime(?y, ?z) \wedge$$
$$swrlb : lessThanOrEqual(?w, ?z) \implies before(?x, ?y)$$

O resultado da atividade dedutiva realizada pelo motor de inferência será atribuir um indivíduo da classe *TV* à classe *WatchTV*, o que significa que este indivíduo representa o intervalo de tempo em que o evento ver televisão ocorreu. O racional disto é não fazer sentido dizer que alguém está a ver televisão se esta não estiver ligada, ainda que esteja na sala e sentado, porém, de igual forma, apenas o facto de a televisão estar ligada não significa que esteja alguém a vê-la, sendo necessário que haja alguém na sala e sentado. Pode-se argumentar que esta não é a única forma de se assistir televisão, uma vez que o utilizador pode entrar na sala, ligar o aparelho e por fim sentar-se na poltrona, ou mesmo, a televisão estar previamente ligada e só depois é que o utilizador entra na sala e se senta na poltrona. Para tanto, basta que se façam mais duas regras que satisfaçam estas condições, visto que o SWRL não admite de raiz a inclusão de operadores de disjunção nas regras.

### 4.4.5 Manipulação do modelo OWL

O modelo OWL representado pela figura 4.15 será o ponto de partida para a criação de instâncias com o Jena. A proposta do módulo é possibilitar a representação de instâncias de eventos relevantes para a caracterização de uma atividade complexa, em termos de composição de múltiplas atividades simples. A vista geral, do modo como esta tarefa é realizada, está representada pela figura 4.16.

Assim, o módulo de representação do conhecimento receberá informações vindas do módulo de automação acerca de eventos de atividades simples, localização e utilização de dispositivos. Esta transmissão de dados dá-se por pedidos HTTP e o controlador do módulo manda que o modelo de dados OWL crie novas instâncias dos eventos. Logo em seguida, o modelo requisita ao motor de inferência para realizar novas inferências e o

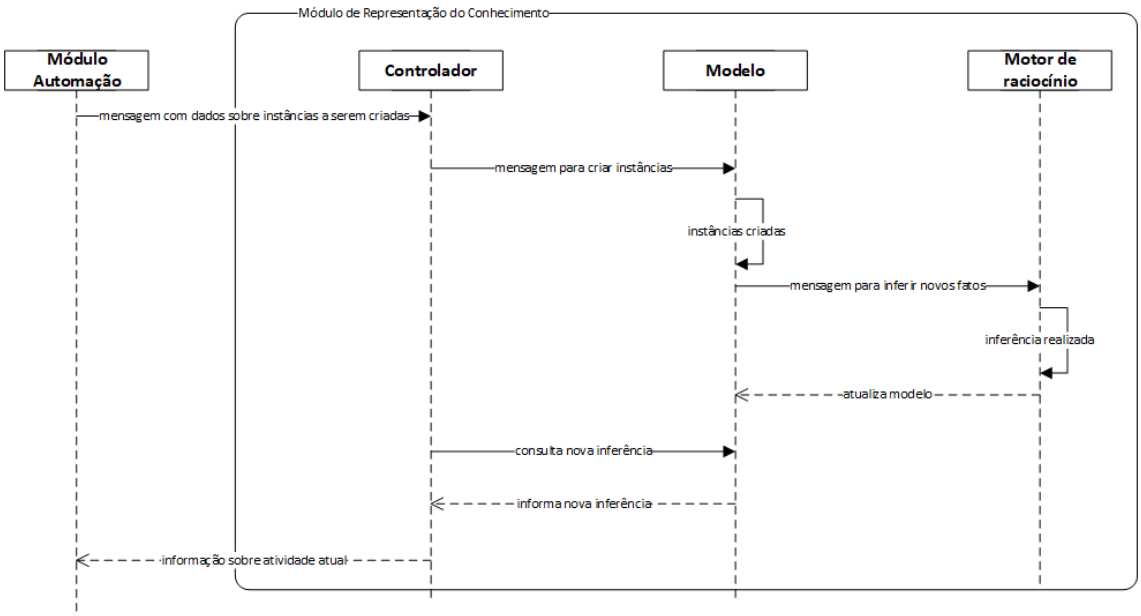



Figura 4.16: *Vista geral do funcionamento do módulo de representação de conhecimento*

modelo estará atualizado com estas informações. A seguir à criação das instâncias o próprio controlador requisita ao modelo que informe sobre a existência de novas inferências. Após esta informação, o controlador envia uma resposta HTTP ao módulo de automação para que atualize a atividade atual do utilizador. Os pormenores de implementação são os que se seguem.

Como já visto na figura 4.14, o Jena é formado por vários componentes, pelo que se destacam o Ontology API e o Reasoner API. O primeiro é composto por diversas classes que se prestam à manipulação de modelos OWL. Como visto, um modelo possui classes, estas possuem instâncias e as relações entre classes são representadas por propriedades. Há propriedades que representam o relacionamento entre uma classe e um tipo de dados. A primeira tarefa do Jena é criar uma instância da classe **OntModel**, que representa o modelo OWL. Isto é feito por meio do método **createOntologyModel** da classe **ModelFactory** e pode-se passar como argumento o motor de inferência a ser utilizado. Por fim, faz-se a instância da classe **OntModel** ler o **InputStream** que contém o conteúdo do ficheiro com o modelo criado pelo Protégé. A figura 4.17 representa as tarefas acima descritas.

```
String file="model.owl";
OntModel ontModel=ModelFactory.createOntologyModel(PelletReasonerFactory.THE_SPEC);
InputStream in= new FileInput(file);
ontModel.read(in,"RDF/XML");
```

Figura 4.17: *Exemplo de como carregar um modelo utilizando o Jena*

A figura 4.18 representa as operações básicas de um modelo OWL. Uma ontologia OWL pode importar outras ontologias de modo a utilizar o modelo por elas representado. Assim, cada ontologia possui um **namespace** próprio capaz de identificá-la. Da mesma forma, o Jena identifica as entidades de um modelo com base no **namespace** da ontologia da qual estas façam parte.
A classe OntModel possui métodos para obter classes (**getOntClass**()), relacionamento entre instâncias (**getObjectProperty**()), relacionamento entre uma instância e um tipo de dados (**getDatatypeProperty**()), bem como criar instâncias (**createIndividual**()) e um **Literal** com um valor em um certo tipo de dado (**createTypedLiteral**()). Uma instância pode adicionar relacionamentos como sendo do tipo **ObjectProperty** ou **DatatypeProperty**, passando-se, respectivamente, um **Individual** ou um **Literal**.

```
OntClass OntClass=ontModel.getOntClass(ontologyNamespace+className);

Individual individual1=ontModel.createIndividual(ontologyNamespace,ontClass);
Individual individual2=ontModel.createIndividual(ontologyNamespace,ontClass);

ObjectProperty objectProperty=ontModel.getObjectProperty(ontologyNamespace+objectPropertyName);

DatatypeProperty datatypeProperty=ontModel.getDatatypeProperty(ontologyNamespace+datatypePropertyName);
Literal literal=ResourceFactory.createTypedLiteral(value,XSDDatatype.XSDdecimal);

individual1.addProperty(objectProperty,individual2);
individual1.addProperty(datatypeProperty,literal);
```

Figura 4.18: *Operações básicas sobre um modelo OWL*

A tarefa de inferir novos factos é feita automaticamente pelo motor de inferência passado para o modelo, não sendo necessária qualquer intervenção do utilizador. Para consultar a lista de instâncias de uma classe inferida, basta obter a classe por meio do **OntModel**, conforme visto acima, e utilizar o método **listInstances**(), que retorna um iterador. A figura 4.19 demonstra o exposto.

```
OntClass inferredClass=ontModel.getOntClass(ontologyNamespace+inferredClassName);
ExtendedIterator<Individual> it=(ExtendedIterator<Individual>) inferredClass.listInstances();
while(it.hasNext()){
    System.out.println(it.next());
}
```

Figura 4.19: *Obter e iterar uma classe inferida*

O Jena ainda possibilita a criação de **Listeners** para permitir que uma ação seja desempenhada quando o modelo for alterado. Para tanto, é necessário que a classe implemente a interface **ModelChangedListener** e inclua o trecho de código a ser executado em um dos métodos constantes do contrato desta interface. No caso deste trabalho, será necessário implementar apenas o método de assinatura *public void addedStatement(Statement*

*statement)*. O **Listener** será útil para desencadear a comunicação entre este módulo e o de automação quando o motor de inferência identificar uma nova atividade complexa.
A estratégia do desenvolvimento da aplicação que implementa a gestão do modelo OWL é a criação de uma classe Java para cada classe OWL. Cada uma destas classes terá um método de criar o **Individual** para a respectiva instância, de modo a incluir a nova instância no modelo OWL. Um exemplo deste método pode ser visto na figura 4.20.

```
public Individual createIndividual(OntModel ontModel){
    OntClass eventClass = Utils.getOntClass(className);
    OntProperty hasBeginning=null;
    OntProperty hasEnd=null;
    Individual hasBeginningInd=null;
    Individual hasEndInd=null;
    if(validateInstant(this.getHasBeginning())) {
        hasBeginning=Utils.getTimeObjectProperty("hasBeginning");
        hasBeginningInd=Utils.createIndividual(this.getHasBeginning());
    }
    if(validateInstant(this.getHasEnd())){
        hasEnd=Utils.getTimeObjectProperty("hasEnd");
        hasEndInd=Utils.createIndividual(this.getHasEnd());
    }
    Individual newEvent=Utils.createIndividual(Utils.getNs()+this.getEventURI(),eventClass);
    Map<Property,RDFNode> properties=new HashMap<>();
    if(hasBeginning!=null) {
        properties.put(hasBeginning,hasBeginningInd);
    }
    if(hasEnd!=null) {
        properties.put(hasEnd,hasEndInd);
    }
    Utils.addProperties(newEvent,properties);
    return newEvent;
}
```

Figura 4.20: *Método que cria um Individual a partir de uma instância Java*

A classe **Utils** possui diversos métodos acessórios para a manipulação do modelo OWL, com destaque para **getTimeObjectProperty**, que cria uma **ObjectProperty** da ontologia Time-OWL, **addProperties** que adiciona a um **Individual** propriedades contidas num **HashMap** e *getNs()* que retorna o **namespace** da ontologia do trabalho. A classe *LivingRoom* estende a classe Event que possui duas variáveis que representam os instantes de início e fim do evento a ser representado. O que o método **createIndividual()** faz, então, é verificar se a instância Java possui instantes de início e fim, criar, para cada um que possuir, o respectivo **Individual**, obter o **ObjectProperty** adequado, em seguida criar um Individual da classe *LivingRoom*, acrescentar os **ObjectProperties** e **Individuals** dos instantes que existirem ao **HashMap**, adicioná-los ao **Individual** *LivingRoom* e retorná-lo.

## 4.5 Conclusão

O capítulo teve por finalidade expor o modo como os diferentes módulos do sistema foram implementados e tratar de temas relacionados às aplicações e técnicas relacionadas.

A secção destinada ao módulo de automação apresentou a aplicação *OpenHAB*, a sua pilha tecnológica, explicou a importância dos *Items* no contexto da comunicação com dispositivos e outras aplicações, como foi feita a configuração de alguns *addons* utilizados e explicadas algumas regras de automação.
A secção sobre o módulo de aprendizagem apresentou o projeto Weka, como a sua aplicação pode ser útil para fins de testes e desenvolvimento e como a sua API foi utilizada no contexto da implementação deste módulo.
A secção sobre o módulo de representação do conhecimento tratou sobre linguagens relacionadas com este domínio, especificamente as linguagens de ontologia, quais as capacidades e limitações em termos de expressividade e decidibilidade, apresentou as aplicações Protégé e Jena para modelagem e manipulação de ontologias, e os seus respectivos papéis no contexto deste trabalho.

# Capítulo 5

# Testes e Avaliação

## 5.1 Introdução

Este capítulo destina-se a avaliar o sistema implementado, tendo em vista a eficácia do mesmo em satisfazer os casos de usos referidos em 3.5. Cada caso de uso utiliza sensores em um ambiente e os seus valores serão simulados por meio de pedidos HTTP enviados ao módulo de automação. Esta abordagem demonstra uma importante mais valia da aplicação OpenHAB, a de permitir criar rapidamente um ambiente de testes em AAL, que pode ser aproveitado para um ambiente de produção sem grandes modificações a nível de parametrização.
Todos os testes foram realizados num único equipamento que executava as implementações dos três módulos, além de um programa de teste que realizava todos os pedidos HTTP, bem como fazia a verificação dos estados do sistema, com recurso à biblioteca de testes JUnit[1]. As especificações do equipamento são as que constam da tabela 5.1.

Tabela 5.1: *Especificações Máquina de Teste*

| Parâmetro | Valor |
|---|---|
| Memória RAM | 12 Gb |
| Disco | 500Gb SSD |
| Processador | Intel i7-7500U |
| Sistema Operativo | Windows 10 64-bit |

## 5.2 Caso de Uso 1: Controle da toma de medicações

O sistema verifica se houve falha na toma de medicamentos do utilizador. A verificação é feita por meio de consulta ao calendário do utilizador (suportado pelo protocolo CalDav),

[1] `http://junit.org`

um dispensador inteligente e um alarme. O objetivo é fazer com que o o sistema alerte o utilizador para o momento de toma de medicamento por meio de um alarme que será desligado quando for detetada a utilização do dispensador. Se dentro de um determinado período, p. ex. 10 minutos, a toma não ocorra, o alarme será desligado e enviado um alerta por email ao cuidador/médico. A figura 5.1 ilustra o comportamento do sistema para os eventos mencionados.

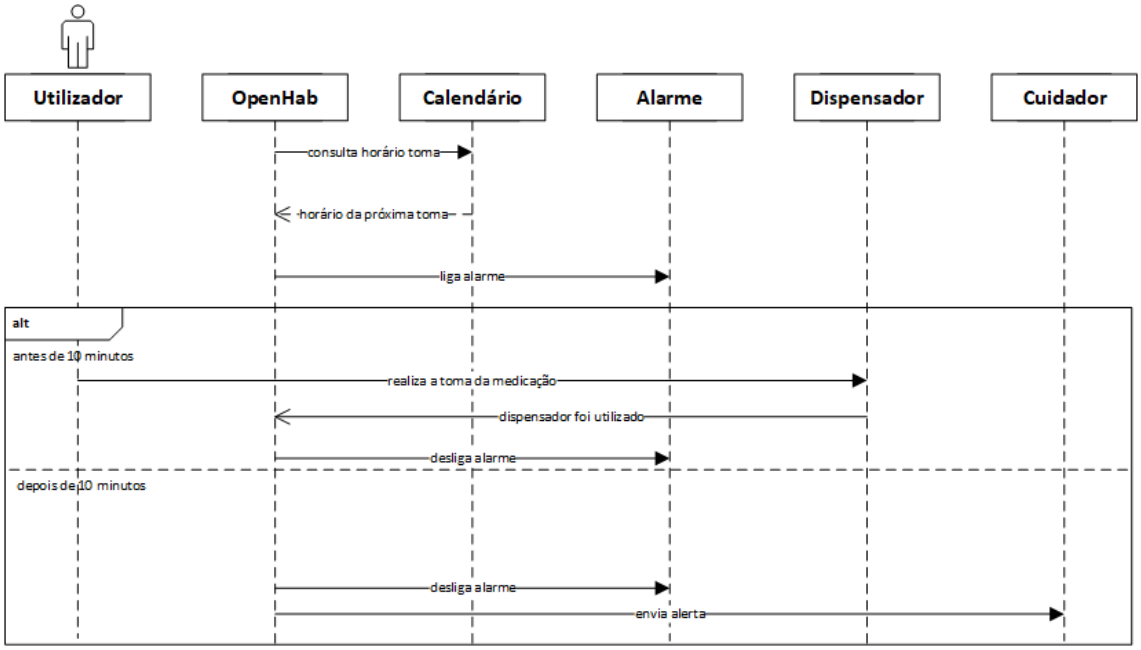



Figura 5.1: *Diagrama de sequência para o Caso de Uso 1*

Foram formuladas duas regras para o OpenHAB, uma que lida com a verificação da próxima toma de medicações e outra que lida com a utilização do dispensador. Uma regra está apenas à espera de que o dispensador seja utilizado para desligar o alarme de toma.
A outra regra realiza algumas tarefas em *polling* a cada minuto. Uma das tarefas é guardar o horário da próxima toma de remédios numa espécie de variável global. A cada minuto é verificado se já está na hora da toma e, nesse caso, é enviado um comando para ativar o alarme. Estando o alarme ativado, o sistema passa a verificar se já decorreram dez minutos, por exemplo, desde o horário em que a toma deveria ter ocorrido. Se o dispensador for utilizado, desliga-se o alarme e guarda-se o horário da próxima toma. Se passou o tempo determinado sem o dispensador ter sido utilizado, significa que o utilizador falhou a toma, então o alarme é desligado e um email é enviado ao cuidador/médico e guarda-se, novamente, o horário da próxima toma.
Dois testes foram implementados para este caso de uso. O primeiro visa determinar que o alarme foi acionado para alertar o utilizador da hora da toma de medicamento, e após um determinado período de tempo, no caso 70 segundos, este usou o dispensador. O

objetivo é determinar se a ordem dos eventos é consistente com o que se espera, ou seja, a determinação de um novo horário de toma da medicação, se já está na hora da toma, o alarme é acionado, o utilizador leva algum tempo até ir ao dispensador, utiliza-o, o alarme é desligado e o sistema fica à espera da nova toma. Pelo fato de a regra possuir como condição um *polling* a cada minuto, foi escolhida uma espera de 70 segundos para ser possível demonstrar que a regra permanece a cada vez que o *polling* é realizado. A imagem 5.2 demonstra que o teste foi bem sucedido, sendo as duas primeiras linhas referentes à determinação de uma nova toma, com a consequente ativação do alarme. Seguindo-se um período de espera, o utilizador ativa o dispensador, para determinar o desligamento do alarme e do próprio dispensador.

```
2017-09-13 11:31:53.322 [INFO ] [runtime.busevents            ] - NextIntake state updated to 2017-09-13T11:31:53
2017-09-13 11:31:53.386 [INFO ] [runtime.busevents            ] - Alarm received command ON
2017-09-13 11:33:03.320 [INFO ] [runtime.busevents            ] - Dispenser state updated to ON
2017-09-13 11:33:03.336 [INFO ] [runtime.busevents            ] - Alarm received command OFF
2017-09-13 11:33:03.350 [INFO ] [runtime.busevents            ] - Dispenser received command OFF
2017-09-13 11:33:03.360 [INFO ] [runtime.busevents            ] - NextIntake state updated to Undefined
```

Figura 5.2: *Resultado do Teste 1*

O segundo teste é parecido com o cenário anterior, exceto que o dispensador não é utilizado. O alarme é ativado durante um minuto e em seguida é desativado e um email é enviado ao cuidador/médico do utilizador, alertando para a falha na toma do medicamento, conforme pode ser verificado na imagem 5.3.

```
2017-09-13 11:34:38.454 [INFO ] [runtime.busevents            ] - NextIntake state updated to 2017-09-13T11:34:38
2017-09-13 11:34:38.507 [INFO ] [runtime.busevents            ] - Alarm received command ON
2017-09-13 11:36:00.005 [INFO ] [org.openhab.model.script.    ] - Email sent to caretaker
2017-09-13 11:36:00.013 [INFO ] [runtime.busevents            ] - Alarm received command OFF
2017-09-13 11:36:00.021 [INFO ] [runtime.busevents            ] - NextIntake state updated to Undefined
```

Figura 5.3: *Resultado do Teste 2*

## 5.3 Caso de Uso 2: Controle de parâmetros do ambiente

Este caso de uso trata do ajuste do ambiente conforme parâmetros determinados pelo utilizador, especificamente, ajustar a luminosidade de uma divisão quando o sistema deteta que o televisor foi ligado. O módulo de automação recebe diversas informações vindas dos sensores e das aplicações, a saber, presença numa divisão, postura do utilizador, estado de dispositivo, envia-os ao módulo de representação do conhecimento para que este infira uma atividade complexa e, conforme qual seja esta atividade, o módulo de automação ajusta os parâmetros do ambiente. O sistema é capaz, também, de obter a informação sobre a luminosidade do ambiente por meio de um sensor e atuar na iluminação . A televisão foi substituída pela aplicação de *media center* Kodi, de modo que é possivel utilizar

as funcionalidades da sua API JSON-RPC[1] para inqueri-la acerca do conteúdo multimédia em execução no momento. Conforme já mencionado, todos os valores dos *Items* do OpenHAB foram, para fins de testes, simulados por pedidos HTTP. O fluxo de dados do sistema é apresentado na figura 5.4.

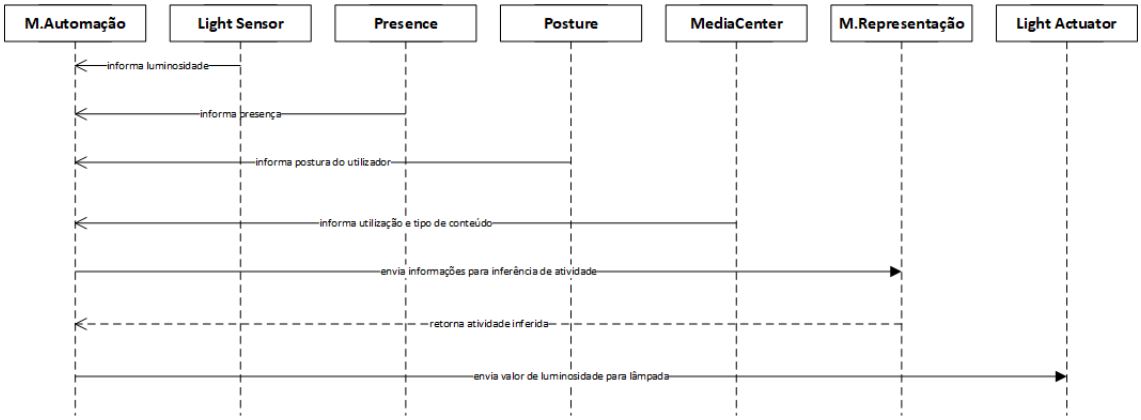



Figura 5.4: *Diagrama de sequência para o Caso de Uso 2*

O cenário de teste foi definido para que o sistema seja capaz de detetar que o utilizador está a ver um filme e fazer com que a luminosidade do ambiente diminua. Este caso de uso exige um conjunto de regras no módulo de automação para lidar com as informações relevantes oriundas do ambiente, a saber, aquelas acerca da postura do utilizador, bem como a sua localização no momento e dispositivo atualmente em uso. Assim, foram criados três itens que guardam estas informações e são modificados sempre que há uma mudança na postura, na localização ou no dispositivo. Cada sensor que detete presença e cada dispositivo que seja utilizado irá modificar o *Item* correspondente. Cada modificação nestes três *Items* executa uma nova regra que envia a informação correspondente ao módulo de representação de conhecimento e verifica se uma nova inferência é realizada após esta nova inclusão. Se houver inferência de uma nova atividade complexa, verifica-se se esta corresponde a assistir a um filme no *Media Center* e, sendo verdade, ajusta os parâmetros de luminosidade para aqueles definidos pelo utilizador.

O primeiro teste realizado visou confirmar se os módulos de automação e representação do conhecimento interagem de forma consistente, de modo a que uma atividade complexa seja inferida adequadamente. Assim, o cliente de teste envia diversos pedidos HTTP ao módulo de automação, nos quais constam informações acerca da entrada do utilizador na sala de estar, em seguida senta-se e, por fim, liga o *media center*. O módulo de automação terá os seus diferentes *Items* modificados e para cada um deles, envia a informação ao módulo de representação. Quando uma atividade complexa for inferida, esta informação será enviada do módulo de representação do conhecimento ao módulo de automação. A

[1] `http://kodi.wiki/view/JSON-RPC_API`

figura 5.5 demonstra os pedidos que chegam ao módulo de automação.

```
2017-09-19 11:43:34.799 [INFO ] [runtime.busevents                ] - LivingRoomSensor state updated to ON
2017-09-19 11:43:34.813 [INFO ] [runtime.busevents                ] - BathroomSensor state updated to OFF
2017-09-19 11:43:34.831 [INFO ] [runtime.busevents                ] - ActualLocalization state updated to Living Room
2017-09-19 11:43:39.804 [INFO ] [runtime.busevents                ] - ActivityDetector state updated to 5
2017-09-19 11:43:39.810 [INFO ] [runtime.busevents                ] - ActualPosture state updated to Sitting
2017-09-19 11:43:44.809 [INFO ] [runtime.busevents                ] - MediaCenter state updated to ON
2017-09-19 11:43:44.820 [INFO ] [runtime.busevents                ] - ActualDevice state updated to MediaCenter
2017-09-19 11:43:46.061 [INFO ] [runtime.busevents                ] - ActualActivity state updated to WatchTV
```

Figura 5.5: *Resultado do teste 1*

Assim, vê-se que o utilizador entrou na sala de estar, ativando o *Item* respectivo. Para fins de consistência do sistema, optou-se por mandar comandos de desligar todos os outros sensores envolvidos, representados no caso pelo sensor da casa de banho. O módulo de automação atualiza o *Item* correspondente à localização atual do utilizador para a sala de estar, no caso *ActualLocalization*. Em seguida, o sistema reconhece que o utilizador adotou a postura de sentar e guarda a informação de postura atual no *Item ActualPosture*. Por fim, o sistema identifica que o *media center* foi ligado e guarda a informação de dispositivo atual no *Item ActualDevice*. Para cada alteração nos *Items* de localização, postura e dispositivo atuais, o módulo de automação envia o *timestamp* do evento para que o módulo de representação do conhecimento crie uma nova instância da classe respectiva. Em seguida o módulo de automação questiona o módulo de representação do conhecimento sobre uma nova inferência e, em caso positivo, este manda um comando ao módulo de automação para que atualize a informação sobre uma atividade complexa. No caso do teste, o módulo de automação atualiza o *Item* de atividade *ActualActivity* para *WatchTV*. O módulo de representação do conhecimento possui a superclasse *WatchTV* e a sua subclasse *WatchTVMovie*. A diferença entre ambas é que a regra para a subclasse envolve verificar se o valor da propriedade *content* na instância *MediaCenter* é "movie", que é enviada pelo módulo de automação após consultar a API *JSON-RPC* do *Kodi*. O objetivo do segundo teste, então, é integrar esta nova informação e fazer com que o módulo de representação infira uma nova instância de *WatchTVMovie*, envie esta informação ao módulo de automação e este envie um comando à lâmpada inteligente para que diminua a sua intensidade. O resultado do teste pode ser verificado na figura 5.6.

## 5.4 Caso de Uso 3: Acompanhamento dos parâmetros cardíacos

Este caso de uso demonstra a utilização concertada entre os diferentes módulos do sistema. Possui dois objetivos principais. Primeiramente visa detetar atividades complexas com base em dados de sensores enviados ao módulo de representação do conhecimento.

```
2017-09-24 10:36:56.383 [INFO ] [runtime.busevents            ] - LivingRoomSensor state updated to ON
2017-09-24 10:36:56.395 [INFO ] [runtime.busevents            ] - BathroomSensor state updated to OFF
2017-09-24 10:36:56.401 [INFO ] [runtime.busevents            ] - ActualLocalization state updated to Living Room
2017-09-24 10:37:01.385 [INFO ] [runtime.busevents            ] - ActivityDetector state updated to 5
2017-09-24 10:37:01.390 [INFO ] [runtime.busevents            ] - ActualPosture state updated to Sitting
2017-09-24 10:37:06.390 [INFO ] [runtime.busevents            ] - MediaCenterContent state updated to movie
2017-09-24 10:37:06.394 [INFO ] [runtime.busevents            ] - MediaCenter state updated to ON
2017-09-24 10:37:06.402 [INFO ] [runtime.busevents            ] - ActualDevice state updated to MediaCenter
2017-09-24 10:37:07.039 [INFO ] [runtime.busevents            ] - ActualActivity state updated to WatchTVMovie
2017-09-24 10:37:07.066 [INFO ] [runtime.busevents            ] - RoomLum state updated to 30
```

Figura 5.6: *Resultado do teste 2*

O segundo objetivo é alimentar o módulo de aprendizagem com a informação da atividade complexa desempenhada pelo utilizador em conjunto com os dados decorrentes dos batimentos cardíacos de modo a detetar se, conforme a atividade complexa atual, os parâmetros cardíacos implicam ou não em uma situação de anormalidade. Em caso positivo, o módulo de automação enviará um alerta para o cuidador/médico do utilizador. A figura 5.7 demonstra as interações entre os diferentes módulos do sistema.

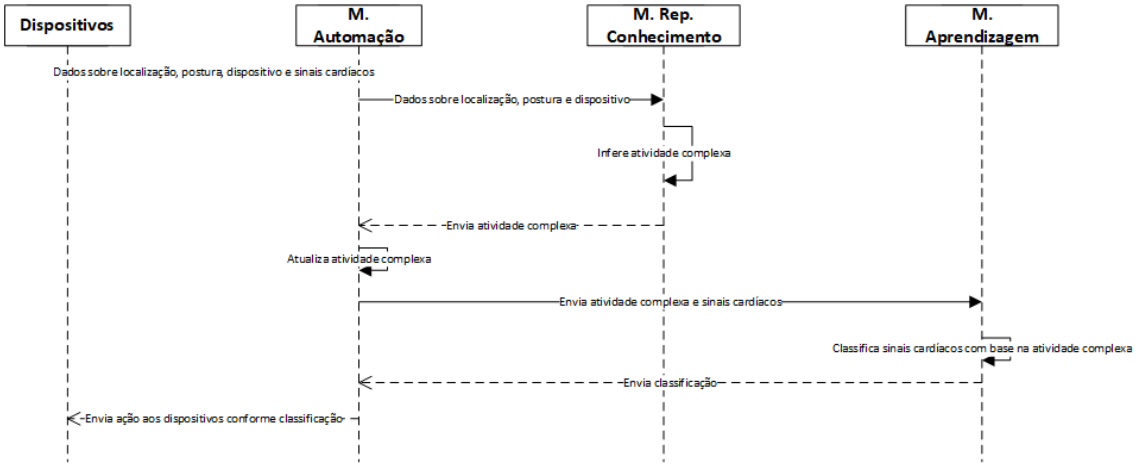



Figura 5.7: *Diagrama de sequência para o caso de uso UC3*

O módulo de automação recebe informações vindas dos sensores existentes na rede e envia comandos ao módulo de representação do conhecimento para que este crie as instâncias adequadas ao modelo de dados da ontologia. Em seguida este módulo realiza dedução de novos fatos e envia a informação de volta ao módulo de automação que atualiza o estado do utilizador. Todas estas tarefas são idênticas àquelas realizadas no teste anterior. O módulo de automação, então, envia a nova informação, e os dados de padrões cardíacos, ao módulo de aprendizagem que realiza a classificação, responde ao módulo de automação, que reage adequadamente a esta classificação. Os dados dos padrões cardíacos são a média dos valores dos batimentos cardíacos, o valor mínimo, o valor máximo e a variância. O módulo de automação recebe uma amostra por segundo e a cada minuto realiza o envio destes valores. A reação, em caso de anormalidade, pode ser um alerta ao

cuidador do utilizador.
O módulo de aprendizagem possui um modelo de classificação apto a aceitar os valores de atividade e padrões cardíacos. O módulo de automação envia os valores em formato JSON, como um vetor de decimais, sendo tratados pelo modelo já treinado do módulo de aprendizagem. O modelo foi treinado com valores reais de normalidade e valores sintéticos de anormalidade, para simular situações de taquicardia do utilizador. A tarefa do modelo é relacionar os valores dos padrões cardíacos do utilizador com a atividade desempenhada e determinar se, para uma mesma atividade, batimentos cardíacos distintos dos padrões de normalidade podem ser considerados como anormais e o sistema ser capaz de gerar alertas nestas situações.
Os valores do utilizador foram obtidos com os dispositivos mencionados em 4.2.2, especificamente a Banda ECG, a placa HRMI e a placa Arduíno. Os valores sintéticos (*Value*) foram criados programaticamente tendo como parâmetros o valor médio que se espera para uma situação de taquicardia (*Mean*), um valor de desvio padrão (*Std*) e um valor aleatório a partir de uma distribuição Gaussiana (*Gaussian*), conforme a fórmula 5.1 abaixo:

$$Value = Gaussian * Std + Mean \tag{5.1}$$

As amostras sintéticas e reais foram incluídas no ficheiro .arff que é utilizado para alimentar o modelo quando o módulo de aprendizagem arranca, já configurado com um meta classificador *CostSensitiveClassifier* que executa o classificador *NaiveBayes* com uma matriz de custo de 10 para os falsos positivos, conforme mencionado na seção 4.3. Os resultados da classificação podem ser vistos nas figuras 5.8a e 5.8b.
O segundo teste envolve a comunicação entre todos os módulos do sistema e consiste em simular um utilizador assistindo televisão, tal qual o caso de uso 2, e em seguida serão fornecidos ao sistema valores de batimentos cardíacos normais e depois anormais para verificar se a classificação ocorre como se espera. O módulo de automação recebe as informações da atividade e dos padrões cardíacos e envia ao módulo de aprendizagem que lhe responderá com a classificação e, em caso de anormalidade, faz com que o módulo de automação envie um email ao cuidador/médico acerca da possibilidade de uma situação de emergência. Os resultados dos testes constam das figuras 5.9a e 5.9b.

## 5.5 Discussão

O racional da escolha dos casos de uso esteve relacionado aos graus de interação que os módulos deveriam possuir, bem como à gradação de complexidade das tarefas eleitas. Assim, primeiramente verificou-se que o módulo de automação é suficiente para realizar tarefas que tenham o padrão de um agente reflexivo baseado em modelo, ou seja, que possui um modelo interno, ainda que rudimentar, do mundo e atua sobre ele de forma

```
Correctly Classified Instances         263               98.1343 %
Incorrectly Classified Instances         5                1.8657 %
Kappa statistic                          0.939
Mean absolute error                      0.0191
Root mean squared error                  0.1104
Relative absolute error                  6.2672 %
Root relative squared error             28.3336 %
Total Number of Instances              268

=== Confusion Matrix ===

   a   b   <-- classified as
 215   3 |   a = normal
   2  48 |   b = abnormal

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0.986    0.040    0.991      0.986    0.989      0.939    0.999     1.000     normal
               0.960    0.014    0.941      0.960    0.950      0.939    0.999     0.996     abnormal
Weighted Avg.  0.981    0.035    0.982      0.981    0.981      0.939    0.999     0.999
```

(a) *Resultados da classificação com matriz de custo unitária*

```
Correctly Classified Instances         261               97.3881 %
Incorrectly Classified Instances         7                2.6119 %
Kappa statistic                          0.9183
Mean absolute error                      0.0322
Root mean squared error                  0.143
Relative absolute error                 10.5633 %
Root relative squared error             36.7136 %
Total Number of Instances              268

=== Confusion Matrix ===

   a   b   <-- classified as
 211   7 |   a = normal
   0  50 |   b = abnormal

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0.968    0.000    1.000      0.968    0.984      0.921    0.999     1.000     normal
               1.000    0.032    0.877      1.000    0.935      0.921    0.999     0.996     abnormal
Weighted Avg.  0.974    0.006    0.977      0.974    0.975      0.921    0.999     0.999
```

(b) *Resultados da classificação com matriz de custo 10 vezes maior sobre os falsos positivos*

```
2017-10-04 13:16:11.305 [INFO ] [runtime.busevents             ] - LivingRoomSensor state updated to ON
2017-10-04 13:16:11.330 [INFO ] [runtime.busevents             ] - ActualLocalization state updated to Living Room
2017-10-04 13:16:16.260 [INFO ] [runtime.busevents             ] - ActivityDetector state updated to 5
2017-10-04 13:16:16.309 [INFO ] [runtime.busevents             ] - ActualPosture state updated to Sit
2017-10-04 13:16:21.264 [INFO ] [runtime.busevents             ] - MediaCenter state updated to ON
2017-10-04 13:16:21.281 [INFO ] [runtime.busevents             ] - ActualDevice state updated to MediaCenter
2017-10-04 13:16:26.414 [INFO ] [runtime.busevents             ] - ActualActivity state updated to WatchTV
2017-10-04 13:16:26.465 [INFO ] [runtime.busevents             ] - HRDetector state updated to 74.43934892027168
2017-10-04 13:16:27.441 [INFO ] [runtime.busevents             ] - HRDetector state updated to 87.58975254001513
2017-10-04 13:16:28.444 [INFO ] [runtime.busevents             ] - HRDetector state updated to 72.27810484713311
2017-10-04 13:16:29.449 [INFO ] [runtime.busevents             ] - HRDetector state updated to 80.8389400691195
2017-10-04 13:16:30.449 [INFO ] [runtime.busevents             ] - HRDetector state updated to 83.38419713405284
2017-10-04 13:16:30.568 [INFO ] [org.openhab.model.script.     ] - normal
2017-10-04 13:16:30.576 [INFO ] [org.openhab.model.script.     ] - Not an emergency
```

(a) *Resultados do teste com padrões cardíacos normais*

```
2017-10-04 13:17:51.226 [INFO ] [runtime.busevents             ] - LivingRoomSensor state updated to ON
2017-10-04 13:17:51.331 [INFO ] [runtime.busevents             ] - ActualLocalization state updated to Living Room
2017-10-04 13:17:56.169 [INFO ] [runtime.busevents             ] - ActivityDetector state updated to 5
2017-10-04 13:17:56.276 [INFO ] [runtime.busevents             ] - ActualPosture state updated to Sit
2017-10-04 13:18:01.172 [INFO ] [runtime.busevents             ] - MediaCenter state updated to ON
2017-10-04 13:18:01.182 [INFO ] [runtime.busevents             ] - ActualDevice state updated to MediaCenter
2017-10-04 13:18:06.311 [INFO ] [runtime.busevents             ] - ActualActivity state updated to WatchTV
2017-10-04 13:18:06.332 [INFO ] [runtime.busevents             ] - HRDetector state updated to 119.64598564718369
2017-10-04 13:18:07.334 [INFO ] [runtime.busevents             ] - HRDetector state updated to 108.67363400224326
2017-10-04 13:18:08.341 [INFO ] [runtime.busevents             ] - HRDetector state updated to 125.01616562088344
2017-10-04 13:18:09.342 [INFO ] [runtime.busevents             ] - HRDetector state updated to 112.48082826479465
2017-10-04 13:18:10.346 [INFO ] [runtime.busevents             ] - HRDetector state updated to 108.82626527108306
2017-10-04 13:18:10.467 [INFO ] [org.openhab.model.script.     ] - abnormal
2017-10-04 13:18:10.473 [INFO ] [org.openhab.model.script.     ] - Emergency, email sent to caretaker
```

(b) *Resultados do teste com padrões cardíacos anormais*

determinística. O segundo caso de uso demonstra que os módulos de automação e representação do conhecimento podem interagir de forma consistente, o que corresponderia a um padrão de agente baseado em conhecimento, em que o módulo de automação adicionaria novas sentenças ao vocabulário do módulo de representação, este poderia inferir novas sentenças a partir destas adições e influenciar as decisões tomadas pelo módulo de

automação. O último caso de uso integrou os três módulos, de modo que alguns dados do módulo de automação seriam processados pelo módulo de representação do conhecimento, devolvidos àquele que os enviaria, juntamente a outros dados, ao módulo de aprendizagem. Esta interação entre os módulos possui um padrão de agente baseado em conhecimento, mas também, de agente baseado em aprendizagem, visto que o módulo com este nome implementa algoritmos de aprendizagem máquina, especificamente aqueles de classificação.

Assim, os testes necessários para o caso de uso "Controle da toma de medicações" verificaram as duas situações possíveis no contexto da toma de medicações, que são quando o utilizador cumpre a tarefa, acionando o dispensador, o alarme parando de tocar; e a segunda situação que é quando o utilizador falha a toma, na qual o alarme também para de tocar, mas um email é acionado para alertar o cuidador do ocorrido.

Para o contexto do caso de uso "Controle de parâmetros do ambiente" o objetivo dos testes relaciona-se com a realização bem sucedida da inferência de atividades complexas com base em eventos realizados pelo utilizador e a atuação do sistema no ambiente conforme o resultado desta inferência. O caso de uso tem como objetivo fazer com que o sistema diminua a luminosidade do ambiente para 30% se o utilizador estiver a ver um filme ou restaurar a luminosidade do ambiente se este estiver a realizar outra atividade, de modo que é necessária a cooperação entre os módulos de automação e de representação do conhecimento.

Por último, o caso de uso "Acompanhamento dos parâmetros cardíacos" integra todos os módulos do sistema para realizar a classificação de padrões cardíacos do utilizador em conjunto com a atividade complexa que esteja a desempenhar, de modo a relacionar estas duas informações. O teste consiste, portanto, em simular o utilizador a ver televisão e depois enviar ao módulo de automação dados simulados sobre os seus padrões cardíacos, primeiro que sejam condizentes com uma situação de normalidade e em seguida dados de anormalidade. O módulo de automação envia a informação sobre a atividade e padrões cardíacos ao módulo de aprendizagem, que irá realizar a classificação e responder ao módulo de automação que, em caso de anormalidade, envia um alerta ao cuidador do utilizador.

Fica, portanto, demonstrado que o sistema se comportou conforme fora projetado, cumprindo, assim, as tarefas que lhe foram atribuídas, na forma dos casos de usos especificados.

# Capítulo 6

# Conclusão

## 6.1 Resultados

A revisão bibliográfica realizada no Capítulo 2 permitiu o aprofundamento em diversos temas. Na seção sobre AAL foi possível demonstrar a relevância do tema, seja em termos de impacto na vida das pessoas, seja no montante de recursos disponibilizados por instituições governamentais e supra-governamentais. Na seção sobre IoT, verificou-se as consequências oriundas da popularização de dispositivos com capacidades de comunicação e as dificuldades enfrentadas pela diversidade de protocolos, o que motivou o desenvolvimento de aplicações de automação domésticas, algumas das quais aqui analisadas. O tópico sobre IA adotou a definição de ramo que estuda os agentes inteligentes e delineou três classes destes agentes para, em seguida, aprofundar mais as discussões sobre os agentes baseados em conhecimento e os agentes baseados em aprendizagem.
A especificação realizada no Capítulo 3 concebeu a arquitetura do sistema em três módulos com capacidades de comunicação baseado em HTTP. A conceção do módulo de automação previu a existência de um motor de regras e a possibilidade de interagir com dispositivos independentemente do protocolo que possam usar, sem o custo de lidar com esta diversidade. O módulo de representação do conhecimento apresentou a necessidade de um modelo de dados para a representação das classes e instâncias, bem como a importância de manipulação deste modelo de dados para a criação de novas instância e, finalmente, a existência de um motor de inferência capaz de derivar novas informações a partir das já existentes. O módulo de aprendizagem estipulou a existência de uma entidade capaz de realizar o processamento dos dados de modo a criar um modelo útil para realizar as tarefas de classificação exigidas pelo sistema. Finalizou-se o capítulo com a especificação dos requisitos funcionais e dos casos de uso que demonstram a validade do sistema.
No Capítulo 4 detalhou-se como o sistema foi implementado em seus diferentes módulos, bem como justificada a escolha das ferramentas utilizadas. Foi apresentada a aplicação de automação que satisfez os requisitos elencados pela especificação, fez-se uma exposição sobre suas mais valias, parametrização e apresentou-se algumas das regras utilizadas.

Na secção destinada ao módulo de representação do conhecimento foram descritas as linguagens utilizadas, sejam elas de ontologias como o OWL, RDFS e RDF, sejam elas de regras como o SWRL. Apresentou-se, também, o conceito de ontologia e as ferramentas utilizadas para modelação, o Protègè, e manipulação, o Apache Jena. O capítulo foi encerrado com a apresentação do modelo de dados utilizado e exemplos de manipulação implementados em linguagem Java.
O Capítulo 5 apresentou os testes para validação dos três casos de uso apresentados no Capítulo 3, discutiu-se a adequação dos mesmos e os seus resultados. Mencionou-se também, quais os módulos envolvidos em cada um dos testes e foram explicados os fluxos de informação em cada um dos casos.
Cabe, ainda, ressaltar que o presente trabalho é a concretização de esforços empreendidos tanto do ponto de vista teórico como do prático. Quanto ao aspecto teórico podem ser mencionados:

- a proposta de arquitetura para um sistema inteligente, dividido em módulos interdependentes com a cooperação de duas técnicas de IA;
- a reflexão acerca de conceitos sobre AAL, IoT e IA, recorrendo a conceitos e definições de outras pessoas que se debruçaram sobre estes temas;
- análise comparativa de aplicações de automação, ressaltando pontos fortes e fracos;
- análise de trabalhos de outros investigadores que utilizaram AAL como contexto e técnicas de IA.

Quanto ao aspecto prático podem ser mencionados:

- experimentação de diferentes aplicações de automação;
- instalação e configuração da aplicação escolhida, bem como criação de entidades, regras de automação e da comunicação de dispositivos físicos, superando as questões de sintaxe e lógica das linguagens de programção internas;
- experimentação de aplicações para modelagem e manipulação de ontologias OWL;
- criação da taxonomia e das regras SWRL, bem como reutilização de outras ontologias já implementadas;
- exploração da API Jena e implementação do código para o módulo de representação do conhecimento;
- desenvolvimento do código, utilizando a biblioteca Weka, para o módulo de aprendizagem.

## 6.2 Trabalho futuro

O sistema implementado corresponde a um protótipo e à concretização de uma proposta de arquitetura. Ainda assim, há diversas limitações que podem ser trabalhadas para o aperfeiçoamento do sistema. O módulo de automação possui limitações no modo como as regras foram implementadas, muitas delas aplicam *polling* na realização das tarefas, o que implica em sobrecarga no sistema. Há, portanto, espaço para melhorar a implementação destas regras com a utilização de regras com condições baseadas em *push*. O módulo de representação do conhecimento exige, em suas regras SWRL, que todo evento seja constituído por início e final. A limitação que isto causa é a constante criação de instância da classe *Instant* para representar um final artificial para eventos em andamento. A consequência é a existência de muitas destas instâncias artificiais sobrecarregando o motor de inferência que tem de lidar com mais instâncias do que as que seriam necessárias, impactando no tempo de resposta do módulo.

Em sede de trabalho futuro, prevê-se uma investigação mais aprofundada de técnicas e aplicações que melhorem o desempenho do módulo de representação do conhecimento, seja no tocante à implementação do programa de manipulação do modelo OWL, de modo a tornar o código mais fácil de manter, seja na elaboração de regras SWRL adicionais. Exemplos de refatoração do código seriam a diminuição do uso de métodos estáticos públicos da classe *Utils*, com a substituição por práticas mais adequadas de programação, evitar a criação contínua de instâncias da classe *Instant* para representar o fim dos eventos ainda em execução. Prevê-se também um aprofundamento da investigação de ontologias temporais, exemplos de utilizações por outros investigadores e reutilização de ontologias mais maduras.

Também está prevista uma investigação que reutilize o sistema para identificar padrões de atividades complexas em mais longo prazo. Exemplo de utilização seria a possibilidade de verificar se o utilizador está ou não a ter comportamentos mais sedentários ou menos sociáveis, com base no seu padrão de atividades passadas. Tal informação poderia ser relevante para inferir se há ou não chance de o utilizador estar a sentir-se mais deprimido e alertar o próprio, bem como o seu cuidador ou parentes.

# Referências


J Allen. Maintaining knowledge about temporal intervals. communications of ACM, Vol. 26, No. 11, 832–843. 26(11):832–843, 1983. URL http://scholar.google.com/scholar?q=related:SfE4Y6t68aMJ:scholar.google.com/{&}hl=en{&}num=20{&}as{_}sdt=0,5. 19, 49

Apache Foundation. Jena architecture overview. URL https://jena.apache.org/images/jena-architecture.png. vii, 47

Sotiris Batsakis and Euripides G M Petrakis. Temporal Representation and Reasoning in OWL 2.0. 1:1–5, 2009. URL http://www.semantic-web-journal.net/system/files/swj855.pdf. 20

Rubén Blasco, Álvaro Marco, Roberto Casas, Diego Cirujano, and Richard Picking. A smart kitchen for ambient assisted living. *Sensors (Basel, Switzerland)*, 14(1):1629–1653, 2013. ISSN 14248220. doi: 10.3390/s140101629. 7

L. Chen, C. D. Nugent, and H. Wang. A Knowledge-Driven Approach to Activity Recognition in Smart Homes. *IEEE Transactions on Knowledge and Data Engineering*, 24 (6):961–974, 2012. ISSN 1041-4347. doi: 10.1109/TKDE.2011.51. 19

Domoticz. Domoticz, a. URL https://domoticz.com/. 9

Domoticz. Domoticz-github, b. URL https://github.com/domoticz/domoticz. vii, 10

Eibe Frank, Mark A Hall, and Ian H Witten. The WEKA Workbench. *Morgan Kaufmann, Fourth Edition*, pages 553–571, 2016. URL http://www.cs.waikato.ac.nz/ml/weka/Witten{_}et{_}al{_}2016{_}appendix.pdf. 40

Flavius Frasincar, Viorel Milea, and Uzay Kaymak. TOWL: Integrating time in OWL. *Semantic Web Information Management: A Model-Based Perspective*, pages 225–246, 2010. doi: 10.1007/978-3-642-04329-1_11. 20

P Gonçalves, J M Torres, P Sobral, and R S Moreira. Remote Patient Monitoring in Home Environments. *Porto Portugal*, pages 101–108, 2009. URL http://isus.ufp.pt/wp-content/uploads/mobihealthinf2009.pdf. 39

Jeedom. Jeedom, a. URL https://www.jeedom.com/site/en/. 9

Jeedom. Jeedom-github, b. URL https://github.com/jeedom/core. vii, 10

Lelde Lace, Renars Liepiņš, and Edgars Rencis. Survey on Ontology Languages. *Lecture Notes in Business Information Processing*, 128(SEPTEMBER):70–84, 2012. ISSN 18651348. doi: 10.1007/978-3-642-33281-4. 18

Ruijiao Li, Bowen Lu, and Klaus D. McDonald-Maier. Cognitive assisted living ambient system: a survey. *Digital Communications and Networks*, 1(4):229–252, 2015. ISSN 23528648. doi: 10.1016/j.dcan.2015.10.003. URL http://dx.doi.org/10.1016/j.dcan.2015.10.003. 7

D. Monekosso, F Florez-Revuelta, and P. Remagnino. Ambient Assisted Living [Guest editors' introduction. *IEEE Intelligent Systems*, 30(4):2 – 6, 2015. ISSN 1541-1672. doi: 10.1109/MIS.2015.63. 5

Mark A Musen and the Protégé Team. The Protégé Project: A Look Back and a Look Forward. *AI matters*, 1(4):4–12, jun 2015. ISSN 2372-3483. doi: 10.1145/2757001.2757003. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4883684/. 46

United Nations. World Population Ageing. page 2090, 2009. vii, 6

OpenHAB. OpenHAB, a. URL http://www.openhab.org/. 11

OpenHAB. OpenHAB Architecture Overview, b. URL https://github.com/openhab/openhab1-addons/wiki/images/architecture.png. vii, 32

OpenHAB. OpenHAB-github, c. URL https://github.com/openhab/openhab. vii, 11

D.L. Poole and A.K. Mackworth. Artificial Intelligence - Foundations of Computational Agents. *Artificial Intelligence*, page 661, 2010. ISSN 00043702. doi: 10.1016/j.artint.2010.12.004. URL http://books.google.com/books?hl=en{&}lr={&}id=eALhh{_}tkpv4C{&}oi=fnd{&}pg=PR7{&}dq=Artificial+Intelligence+-+Foundations+of+Computational+Agents{&}ots=NrcQfo1ap1{&}sig=nqtbeObEyGJkcViqG8t6eGK7j8o. 16, 17

Jan S. Rellermeyer, Michael Duller, Ken Gilmer, Damianos Maragkos, Dimitrios Papageorgiou, and Gustavo Alonso. The software fabric for the Internet of things. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4952 LNCS:87–104, 2008. ISSN 03029743. doi: 10.1007/978-3-540-78731-0_6. 8

Stuart Russell and Peter Norvig. *Artifical Intelligence: A Modern Approach (Third Edition)*. 2009. ISBN 9780136042594. doi: 10.1017/S0269888900007724. vii, 12, 13, 14, 15, 17

Artur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959. ISSN 0018-8646. doi: 10.1147/rd.33.0210. 15

Uli Sattler, Robert Stevens, and Phillip Lord. How does a reasoner work? \url{http://ontogenesis.knowledgeblog.org/1486}, 2014. URL http://ontogenesis.knowledgeblog.org/1486. 18

Sandra Sendra, Lorena Parra, Jaime Lloret, and Alejandro Canovas. A Smart Communication Architecture for Ambient Assisted Living. *2014 International Conference on Engineering, Technology and Innovation (ICE)*, (January):1–7, 2015. ISSN 0163-6804. doi: 10.1109/ICE.2014.6871541. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6871541. 7

Thanos G. Stavropoulos, Georgios Meditskos, and Ioannis Kompatsiaris. DemaWare2: Integrating sensors, multimedia and semantic analysis for the ambient care of dementia. *Pervasive and Mobile Computing*, 34:126–145, 2016. ISSN 15741192. doi: 10.1016/j.pmcj.2016.06.006. URL http://dx.doi.org/10.1016/j.pmcj.2016.06.006. 20

Frank van Harmelen. *Handbook of Knowledge Representation*, volume 7. 2015. ISBN 9788578110796. doi: 10.1017/CBO9781107415324.004. 17

W3C. TimeOWL-Allen Figure. URL https://www.w3.org/TR/owl-time/{#}topology. vii, 19

Mark Weiser. The Computer for the Twenty-First Century. *Scientific American*, 265(3):94–104, 1991. ISSN 15591662. doi: 10.1145/329124.329126. URL https://ctools.umich.edu/access/content/group/01ef2b16-f564-4682-8045-14ce56611c49/Readings/Weiser91-ComputerFor21st-SciAm.pdf. 1