Universidade Federal de Ouro Preto
Departamento de Computação e Sistemas
Curso Sistemas de Informação

AIHC - Artificial Intelligence for Home Control

Igor Duarte Rodrigues
igordrodrigues(a)gmail.com

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
PROF. DR. Mateus Ferreira Satler

Agosto, 2017
João Monlevade/MG

Igor Duarte Rodrigues

AIHC- Artificial Intelligence for Home Control

Orientador: Prof. Dr. Mateus Ferreira Satler

Monografia apresentada ao Curso de Sistemas de Informação do Departamento de Computação e Sistemas, como requisito parcial para aprovação na Disciplina Trabalho de Conclusão de Curso II.

Universidade Federal de Ouro Preto
João Monlevade
Agosto de 2017

R696a Rodrigues, Igor Duarte.
AIHC [manuscrito]: Artificial Intelligence for Home Control / Igor Duarte Rodrigues. - 2017.

109f.: il.: color; grafs; tabs.

Orientador: Prof. Dr. Mateus Ferreira Satler.

Monografia (Graduação). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Departamento de Computação e Sistemas de Informação.

1. Sistemas de controle inteligente . 2. Inteligência Artificial . 3. Agentes inteligentes (Software). 4. Automação residencial. 5. Controle eletrônico. I. Satler, Mateus Ferreira. II. Universidade Federal de Ouro Preto. III. Titulo.

CDU: 681.5

Catalogação: ficha@sisbin.ufop.br

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E APLICADAS
COLEGIADO DO CURSO DE SISTEMAS DE INFORMAÇÃO

ATA DE DEFESA

Aos 17 dias do mês de Agosto de 2017, às 10 horas e 40 minutos, na sala H102 do Instituto de Ciências Exatas e Aplicadas, foi realizada a defesa de Monografia pelo aluno **Igor Duarte Rodrigues,** sendo a Comissão Examinadora constituída pelos professores: Prof. Dr. Mateus Ferreira Satler, Prof. Dr. George Henrique Godim da Fonseca e Prof. MSc. Darlan Nunes de Brito.

O candidato apresentou a monografia intitulada: *"AIHC - Artificial Intelligence for Home Control"*. A comissão examinadora deliberou, por unanimidade, pela aprovação do candidato, com nota 9,5 (NOVE PONTOS E MEIO), concedendo lhe o prazo de 15 dias para incorporação das alterações sugeridas ao texto final.

Na forma regulamentar, foi lavrada a presente ata que é assinada pelos membros da Comissão Examinadora e pelo graduando.

João Monlevade, 17 de Agosto de 2017.

Prof. Dr. Mateus Ferreira Satler
Professor Orientador/Presidente

Prof. Dr. George Henrique Godim da Fonseca
Professor Convidado

Prof. MSc. Darlan Nunes de Brito
Professor Convidado

Igor Duarte Rodrigues
Graduando

Curso Sistemas de Informação

TERMO DE RESPONSABILIDADE

O texto do trabalho de conclusão de curso intitulado "AIHC – Artificial Intelligence for Home Control" é de minha inteira responsabilidade. Declaro que não há utilização indevida de texto, material fotográfico, código fonte do programa ou qualquer outro material pertencente a terceiros sem as devidas referências ou consentimento dos referidos autores.

João Monlevade, 17 de agosto de 2017

Igor Duarte Rodrigues

Para Indianara Santos Rodrigues,
Flávio Henrique Freitas Corrêa e
José Airton Marques Junior

Agradecimentos

Agradeço a minha avó Rosaly e meu avô Pedro Delarrue, constante fonte de inspiração. Aos meus pais Rogério e Juliana pelo carinho, compreensão, apoio e os diversos ensinamentos. A meu irmão Victor pela parceria incondicional. Meus tios Humberto, Ana Paula e Sônia pelo apoio e carinho. A meu orientador Prof. Dr. Mateus Ferreira Satler, pela paciência, ensinamentos e as correções que me proporcionaram um aprendizado imensurável do qual serei eternamente grato. Também agradeço através dos professores, Dr. Leonardo Vieira dos Santos Reis, Dr. Rodrigo Geraldo Ribeiro e MSc. Elton Máximo Cardoso, todos os professores que foram fundamentais na minha formação acadêmica. A Icaro Duarte Ribeiro cujas palavras de incentivo a longa data nunca me esquecerei. Ao MSc. Matheus Duarte Brito, Gustavo Endrigo de Sá Fonseca e Bruno Freitas Lage, amigos que me serviram de exemplo. A Jader Felipe Silva Duarte, Breno Brum Ventura e Natan Lima de Ávila, amigos que muito me motivaram durante a produção deste trabalho.

“NO! Try not!
DO or DO NOT, There is no try.”
-Yoda (Star Wars V,1980)

Resumo

O uso da domótica (automação residencial) tem crescido consideravelmente nos últimos anos, principalmente pelo fato de proporcionar comodidade e conforto a seus usuários. Outro campo em destaque é o da Inteligência Artificial. Diversas aplicações têm sido aprimoradas através da aplicação de técnicas de inteligência artificial, abrangendo diversas áreas do saber. Com isso, aplicar técnicas de inteligência artificial para melhorar os sistemas domóticos mostra-se um caminho promissor. Complementarmente, equipamentos que se conectam a internet, classificados no campo da internet das coisas (IoT), vem ganhando destaque e crescendo a cada dia. O grande problema dessa evolução da IoT é que não existe uma padronização definida, o que faz com que equipamentos de fabricantes diferentes não consigam comunicar-se entre si, ocasionando um não aproveitamento total dessa tecnologia. Neste contexto, torna-se importante a existência de um *Framework* domótico que possibilite a criação de uma padronização, sendo crucial que este seja um software livre. Adicionalmente é importante que esse *Framework* seja modular, possibilitando integrações de futuras melhorias. O objetivo deste trabalho é desenvolver um *Framework* modular, integrado com um módulo de inteligência artificial, que além de todas as funções de um sistema domótico, ainda possua a capacidade de auxiliar os usuários a encontrar regras de automação que contribuam com a comodidade dos moradores. Além disso, com um *Framework* de código aberto, pretende-se induzir a padronização da IoT possibilitando assim um melhor aproveitamento de todas essas tecnologias, trabalhando como unidade.

Palavras-Chave: Inteligência Artificial, Domótica, Agente inteligente.

Abstract

The use of domotic (home automation) has grown considerably in recent years, mainly because it provides comfort. Another field in focus is the Artificial Intelligence. Several applications have been improved through the application of artificial intelligence techniques, covering several areas of knowledge. Thus, applying artificial intelligence techniques to improve domotic systems shows is a promising approach. Complementarily, equipment that connects to the Internet, classified in the field “Internet of things” (IoT), has been gaining prominence and growing every day. The major problem with this evolution of IoT is that there is no standard defined, which means that equipment from different manufacturers can not communicate with each other, causing the lack of total benefits of this technology. In this context, it becomes important to have a domotic *Framework* that allows the creation of a standard, being crucial that this is free software. Additionally, it is important that this *Framework* being modular, allowing future improvements to be integrated. The objective of this work is to develop a modular *Framework*, integrated with an artificial intelligence module that, besides all the functions of a home automation system, still has the capacity to help users to find automation rules that contribute to the convenience of the residents. In addition, with an open-source *Framework*, it is intended to induce the standardization of IoT allowing a better use of all these technologies, working as a unit.

Keywords: Artificial Intelligence, Domotic, Intelligent Agent.

Lista de Figuras

Lista de Tabelas

SUMARIO

Sumário

1. INTRODUÇÃO

Duas áreas do conhecimento que têm recebido bastante atenção do meio acadêmico são a domótica e a A.I. (do inglês *Artificial Intelligence*, ou inteligência artificial). A primeira é a ciência que estuda a automação residencial. A segunda é a ciência que busca desenvolver agentes capazes de tomar decisões embasadas no ambiente ao qual estão inseridos com a máxima eficiência possível.

A domótica visa aumentar o conforto e comodidade dos moradores das residências que contam com seus sistemas implantados. Esse conforto e comodidade são advindos das automações criadas em tais sistemas. Uma vez que é possível associar a leitura de um sensor com a mudança de estado de um equipamento, permite o residente criar diversas regras no sistema domótico para que, dada uma determinada leitura de um sensor, o sistema domótico altere automaticamente o estado de um equipamento, por exemplo, quando uma pessoa entrar na sala, a luz acender automaticamente.

De forma geral, os avanços nos estudos do campo da A.I. têm tornado cada dia mais próximo da realidade as ideias apresentadas em filmes de ficção científica. A concepção de uma casa inteligente por exemplo, capaz de perceber as rotinas de seus moradores deduzindo regras de automação para a residência, está cada vez mais próximo de se tornar realidade.

Ao trabalhar a domótica associada à inteligência artificial, começa-se a dar um passo em direção a uma nova realidade para a sociedade. Algo que pode afetar positivamente diversos aspectos de uma comunidade que faça uso dessa tecnologia. Esse conjunto possui um potencial comunitário enorme se utilizada em larga escala como por exemplo economia de energia a nível global. E abre uma nova gama de aplicações para o uso dessa tecnologia, como por exemplo um sistema de vigilância comunitária. Tome como hipótese o caso em que todas as casas possuíssem um sistema deste tipo implantado, associado a um banco de dados fornecido pela polícia, contendo dados biométricos de procurados pela justiça. Isto implicaria que a localização de foragidos seria muito mais rápida e eficiente, resultado de existir um processamento distribuído dessa busca. Cada residência faz a busca em seus sensores e câmeras externas, abrangendo uma área de busca maior em relação ao que é possível quando apenas a polícia se encarregar disso.

Este cenário hipotético depende do uso em larga escala de um sistema domótico inteligente. Por consequência torna-se fundamental a existência de um sistema gratuito e de código aberto, permitindo que um maior número de pessoas tenha acesso a essa tecnologia e que toda uma comunidade trabalhe no seu avanço.

Logo, este trabalho vem propor um *Framework* modular com um módulo de inteligência artificial, que possibilite a utilização e difusão dessa tecnologia. Visando a construção de um benefício comunitário, esse sistema proposto terá seu código aberto e seu uso gratuito, possibilitando maior acessibilidade a uma tecnologia que tem o potencial de trazer benefícios não apenas ao usuário direto, mas para toda a comunidade ao qual ela esteja inserida.

Para o desenvolvimento do presente trabalho foram utilizadas pesquisas bibliográficas e simulações. As pesquisas bibliográficas basearam-se em publicações científicas das áreas de domótica, inteligência artificial, heurística e simulações. As simulações utilizadas tiveram como objetivo validar o modelo proposto sem a necessidade de implantar o sistema em um ambiente físico, o que reduziu consideravelmente o custo de validação.

1.1. Problema

O presente trabalho vem propor um *Framework* domótico integrado com uma inteligência artificial capaz de contribuir com a criação de regras de automação para residência.

1.2. Objetivos

Esse estudo visa alcançar os objetivos apresentados nesta Seção.

1.2.1. Geral

Esse trabalho possui como objetivo geral, desenvolver um sistema que possibilite o controle residencial de qualquer equipamento eletroeletrônico, dentre os quais podemos citar lâmpadas, portões, televisores, rádios, equipamentos de ar-condicionado entre outros; tal como um sistema domótico. Além disso deve possuir uma inteligência artificial para proporcionar maior comodidade para os usuários, através do reconhecimento de padrões de suas atividades, transformando-as em regras de automação. Adicionalmente visando atingir a maior parte possível da sociedade opta-se por um sistema livre e de código aberto.

1.2.2. Objetivos Específicos

Para validar o modelo e garantir que o objetivo geral foi alcançado, apresenta-se alguns objetivos específicos:

- Construção de um *Framework* modular, capaz de controlar eletroeletrônicos presentes em uma residência e monitorar as leituras de sensores instalados na residência.

- Construção de um agente inteligente, que seja integrado ao *Framework* supracitado, que extraia os dados processados gerando sugestões de regras de automação ao usuário.
- Construção de um ambiente virtual para simular uma residência. A finalidade desse ambiente virtual é permitir a validação do *Framework*.
- Construção de um agente de teste para simular a interação de pessoas em uma residência. O intuito desse agente é proporcionar os testes no ambiente virtual, para a validação da inteligência artificial.
- Definição de um conjunto de regras que devem ser descobertas pelo agente inteligente através da observação das rotinas realizadas pelo agente de teste, a fim de validar o modelo global proposto neste trabalho.

1.3. Justificativa

A cada dia surgem novas tecnologias com a finalidade de aumentar o conforto e a segurança das pessoas, das quais muitas são aplicadas no uso domiciliar. Porteiros eletrônicos, geladeiras inteligentes e porta-retratos que se conectam a internet são alguns exemplos tratados pela IoT (Internet of things, ou em português internet das coisas). A domótica, ciência que estuda a automação residencial ganha mais força e se torna mais real a cada dia devido ao avanço tecnológico.

Tem-se também que a inteligência artificial tem avançado consideravelmente em diversas áreas de conhecimento. Portanto associar a domótica a agentes inteligentes potencializa o aumento significativo da comodidade, segurança e economia em relação aos sistemas já existentes, implicando em maior escalabilidade ao uso de sistemas seguindo esse modelo e abrindo uma nova gama possibilidades.

Todavia, o custo de implantação de sistemas domóticos ainda não possibilitam seu uso em larga escala, surgindo como consequência duas necessidades que devem ser sanadas para possibilitar a difusão dessa tecnologia na sociedade. Primeiro, um sistema domótico inteligente, que amplie os ganhos advindos da automação residencial, tornando essa tecnologia e seu uso ainda mais atraentes. Segundo, um sistema livre e de código aberto, que porta dois benefícios diretos: a redução de custo de implantação, uma vez que o software é gratuito e a possibilidade de um trabalho conjunto de toda a comunidade, proporcionado avanços rápidos e significativos, ao exemplo do que ocorre com os sistemas operacionais Linux.

1.4. Estrutura do trabalho

O presente trabalho está dividido em seis capítulos de forma a facilitar a compreensão de seu conteúdo.

O Capítulo 2 apresenta uma revisão da literatura assim como a explicação de alguns conceitos importantes para o entendimento do trabalho. Apresenta uma revisão sobre inteligência artificial, agentes inteligentes e alguns conceitos sobre tecnologias que dão suporte ao sistema que este trabalho propõe; tais como biometria, controle de presença e banco de dados. Apresenta ainda uma definição de ambiente virtual, linguagens de programação com foco em C++ e heurística, destacando uma metaheurística denominada *Ant Colony Optimization* proposta por Dorigo e Stützle (2004).

No Capítulo 3 é feita a formalização do problema e são apresentados os requisitos necessários para alcançar o objetivo proposto. Na sequência é apresentado o modelo utilizado para o sistema, assim como seu método de construção. Ainda a apresentação das fórmulas utilizadas para o funcionamento do sistema.

O Capítulo 4 apresenta a forma como o sistema foi implementado, explicando detalhadamente os principais métodos, classes e funções utilizadas. Separando as implementações do *Framework*, bancos de dados, agente inteligente e ambiente virtual para simulação

O Capítulo 5 apresenta a simulação utilizada para validar o sistema, assim como comentários sobre os resultados obtidos.

Por fim o Capítulo 6 apresenta a conclusão do trabalho realizado e propostas de trabalhos futuros que podem ser realizados como continuidade do aqui apresentado.

2. CONCEITOS GERAIS E REVISÃO DA LITERATURA

Esse Capítulo foi subdividido em seções agrupando técnicas e tecnologias afins. Visto que o grande foco deste trabalho é o uso de inteligência artificial, torna-se interessante começar explicando sobre inteligência artificial e aprofundando em seguida em agentes inteligentes, um dos pilares fundamentais para a construção da A.I. Também neste Capítulo são apresentadas as principais tecnologias utilizadas apoiar a construção do sistema foco deste trabalho. Por fim, é feita uma revisão sobre heurística, técnica utilizada para resolução de problemas do porte do tratado por este estudo.

2.1. Inteligência Artificial

Para definir o conceito de A.I., é preciso trabalhar o conceito de inteligência. Kasabov (1998) define inteligência como a capacidade de aprendizado, de adaptação, tomar decisões apropriadas e comunicar-se através de linguagem, imagens ou de uma forma mais sofisticada.

Essa é uma definição difusas, o que dificulta uma formalização da inteligência e estendendo-se para a Inteligência Artificial, uma vez que para definir o que é algo artificial é preciso definir seu lado natural. Ampliando esse conceito, Kasabov (1998) afirma que a Inteligência Artificial é a área que estuda a resolução de problemas que precisam da inteligência humana. Ele afirma ainda que o principal objetivo da inteligência artificial é desenvolver métodos e ferramentas que permitam a resolução de problemas complexos que exigem inteligência humana para serem resolvidos.

Assim, percebe-se que a definição de inteligência artificial está completamente ligada a definição de inteligência natural (inteligência humana no caso), mas que a tomada de decisões assim como o aprendizado são parte fundamental da A.I. Cujo objetivo é criar sistemas que simulam as faculdades intelectuais humanas.

2.1.1. Agentes Inteligentes

No campo da inteligência artificial, Agentes Inteligentes são programas capazes de, dada uma determinada entrada de dados que constitui um problema, produzir uma saída de dados que corresponda a uma solução do problema proposto. A questão que difere um agente inteligente de um programa será debatida na próxima subseção e em seguida será feita uma classificação dos agentes.

2.1.2. Definição de agente inteligente

A definição de agente inteligente depende diretamente da definição de agente. Segundo Russell e Norvig (1995), um agente é qualquer coisa que percebe e interage com ambiente ao qual está inserido. Eles afirmam que um agente humano possui olhos, orelhas e outros órgãos que funcionam como sensores, assim como mãos e pernas que são seus atuadores. Sendo assim pode-se traçar um paralelo com um agente robótico, que utiliza equipamentos tais como câmeras e motores que equivalem a sensores e atuadores.

Seguindo esse conceito, Russell e Norvig (1995) definem como agente inteligente o conjunto de arquitetura e programa, onde a arquitetura são os sensores e atuadores e o programa é o processamento das entradas para gerar uma saída. Atentando para a existência de agentes completamente virtuais, os autores classificam esses agentes como “software robô ou softbot” (RUSSELL; NORVIG, 1995, p36). Um exemplo de um agente inteligente completamente virtual é um “softbot desenvolvido para pilotar um avião 747 em uma simulação de voo” (RUSSELL; NORVIG, 1995, p36).

Franklin e Graesser (1996) fazem uma análise da definição de agente dada por diversos especialistas. Após essa análise os autores sugerem que um agente inteligente deve possuir um certo conjunto de características dentre as seguintes: i) fazer parte de um ambiente; ii) agir de forma autônoma conforme percebe seu ambiente, iii) ser capaz de alimentar suas entradas de dados sozinho; iii) não necessitar que alguém interprete e/ou use sua saída de dados; iv) ser capaz seguir um cronograma de ação criado por ele; v) alcançar objetivos proposto por outros, seja um agente artificial, um software ou um humano; vi) cada uma de suas ações deve possuir efeito sobre sua percepção futura do ambiente; e vii) enfim, cada uma de suas ações deve ser contínua. Seguindo essa análise, os autores definem um agente inteligente como: um sistema que faz parte de um ambiente, agindo conforme seu próprio cronograma e ao passar do tempo altera sua percepção do ambiente segundo suas experiências.

A análise feita por Franklin & Graesser (1996), corrobora a afirmação de que um agente inteligente é um sistema dotado de sensores e atuadores, capaz de perceber as mudanças do ambiente ao qual está inserido e se adequar a cada novo estado, como também já afirmado por Russell e Norvig (1995).

Ao observar o filme Matrix (1999) tem-se um exemplo claro de Agente Inteligente. Segundo o enredo do filme, os seres humanos estão presos em um sistema de computador, controlado por diversos tipos de agentes, cada um com sua área de atuação. Esse é o mesmo conceito de um agente no campo da A.I.: programas desenvolvidos para executar tarefas e interagir com o ambiente, seja ele real ou virtual.

Dessa forma conclui-se que um Agente Inteligente atua em um determinado ambiente e, conforme esse se altera, o agente se adapta, criando uma nova forma de percebê-lo. Não importa se esse ambiente é virtual como por exemplo um agente jogador de xadrez eletrônico, ou um ambiente real como por exemplo um agente que controla o tráfego do trânsito através de semáforos.

2.1.3. Classificação do Ambiente

O ambiente ao qual o agente está inserido influencia consideravelmente a forma como ele deve ser modelado. Partindo deste princípio Russell e Norvig (1995) classificam o ambiente segundo cinco características: acessibilidade, determinismo, episodicidade, estaticidade e continuidade.

Existem duas possíveis formas de acessibilidade para um ambiente conforme a definição de Russell e Norvig (1995), acessível e inacessível. Os autores definem um ambiente acessível como aquele em que o agente consegue perceber todo o ambiente para tomar uma decisão. Eles exemplificam por um agente jogador de xadrez, que é capaz de perceber a posição de todas as peças no tabuleiro antes de decidir qual jogada efetuar. Caso não seja possível o agente perceber completamente o ambiente, ele é dito inacessível, e pode ser exemplificado como um agente para um jogo de buraco, o qual não possui como definir a mão de um jogador apenas observando o momento atual.

O determinismo do ambiente é igualmente dividido em duas possibilidades. Segundo Russell e Norvig (1995), quando o próximo estado do ambiente depende exclusivamente do estado atual e da decisão do agente, considera-se um ambiente determinístico. Caso existam outros fatores que influenciam no próximo estado do ambiente diz-se que ele é não determinístico. Um ambiente determinístico é aquele em que o agente é capaz de prever todas as mudanças de estado possíveis segundo suas ações e um dado estado inicial.

Em relação a característica episódica de um ambiente, Russell e Norvig (1995) afirmam que um ambiente é episódico quando a experiência do agente se divide em episódios e suas ações se limitam a um período específico, não influenciando para o próximo. Em contrapartida, se o agente precisa pensar a frente para decidir uma sequência de decisões, nas quais uma influencia o resultado da outra, o ambiente é dito não episódico.

A característica de estaticidade do ambiente, é dividida em estático e dinâmico. Segundo Russell e Norvig (1995), um ambiente estático não se altera enquanto o agente aguarda para tomar uma decisão. Caso o ambiente sofra alteração de seu estado enquanto o agente toma sua decisão, diz-se então que ele é dinâmico. Há ainda uma terceira

possibilidade para essa característica, que é do ambiente semi dinâmico. Esse cenário ocorre quando o ambiente não se altera enquanto o agente toma sua decisão mas o tempo de resposta do agente é considerado para sua avaliação de desempenho.

Por último, a quinta característica de um ambiente é se ele é discreto ou contínuo. Russell e Norvig (1995), definem que um ambiente é discreto quando os estados do ambiente são limitados e existe clareza em observar todos os valores das variáveis envolvidas. Eles citam como exemplo o agente jogador de xadrez, que consegue determinar com exatidão a posição de cada peça, assim como quais os possíveis movimentos para cada uma delas. O ambiente é contínuo quando existem variáveis estocásticas, não sendo possível determinar com exatidão e clareza todos os valores em um dado momento.

Observando as definições dadas, diz-se que cada ambiente possui uma combinação dessas características, de forma que irão definir a modelagem a ser utilizada para concepção do agente.

2.1.4. Classificação dos agentes

Agentes inteligentes são desenvolvidos para desempenhar uma série de tarefas diferentes. Para cada tipo de tarefa e conforme sua complexidade uma modelagem é indicada. Russell e Norvig (1995) classificam os agentes em quatro tipos conforme sua modelagem: reflexivo simples, rastreadores do ambiente, baseados em objetivo e baseados em utilidade.

O agente reflexivo é definido por Russell e Norvig (1995) como um agente que se baseia em uma tabela composta por condições e ações, quando uma determinada condição presente na tabela ocorre a ação correspondente é tomada. Esse tipo de agente possui a limitação de apenas decidir coisas já previstas em sua implementação, visto que se baseia exclusivamente na tabela de condição-ação, sendo esse o tipo mais simples de agente.

O agente rastreador de ambiente, segundo os mesmos Russell e Norvig (1995), diferencia-se do primeiro modelo por possuir um controle do estado do ambiente internamente. Apesar de também se basearem em uma tabela de decisão, essa classe de agentes, possui uma memória onde são armazenadas informações importantes sobre o ambiente auxiliando na tomada de decisão.

O agente baseado em objetivo é um pouco mais complexo, ainda segundo Russell e Norvig (1995), essa classe de agente inteligente possui um estado interno de objetivos, o qual é consultado antes de tomar uma decisão. Sendo assim, considera não apenas o estado atual do ambiente e a tabela de possíveis ações, mas também verifica se a ação irá leva-lo a cumprir o objetivo esperado. Em outras palavras, o agente baseado em objetivo faz uma análise da situação atual e verifica qual resultado de cada possível ação, escolhendo sempre uma sequência de ações, que o levam ao seu objetivo.

A última classe de agentes proposta por Russell e Norvig (1995), é a de agentes baseados em utilidade. Essa classe analisa o ambiente e as possíveis sequências de ações. Existindo duas ou mais sequências de ações distintas que o levam a seu objetivo, utiliza uma medida conforme sua programação, verificando qual dessas sequências tem o menor custo, para dessa forma escolher qual sequência de ações tomar.

Além dessas classificações apresentadas por Russell e Norvig (1995), ainda convém considerar outras características na classificação de um agente inteligente. Franklin e Graesser (1996) apresentam uma lista com nove propriedades para classificar um agente: reativo, autônomo, orientado a objetivo, tempo contínuo, aprendizagem, mobilidade, flexibilidade e personalidade.

As definições de Franklin e Graesser (1996) para agente reativo (agente reflexivo) e agente orientado a objetivo são as mesmas utilizadas por Russell e Norvig (1995) para as respectivas classificações. Entretanto as outras propriedades citadas pelos primeiros, propiciam subdividir a classificação de agentes.

A definição dada por Franklin e Graesser (1996) para as outras características são enumeradas e explicadas como: i)agente autônomo: é capaz de controlar suas próprias ações; ii)agente de tempo contínuo: permanece processando por tempo indefinido; iii) agente comunicativo: se comunica com outros agentes, o que pode incluir agentes humanos; iv) agente com aprendizagem: se baseia em experiência anterior com o objetivo de tomar melhores decisões no futuro; v) agente com mobilidade: é capaz de mudar de uma máquina para outra; vi) agente flexível: não tem um roteiro pré-estabelecido para suas ações; e vii) agente com personalidade: possui uma personalidade e um estado emocional.

Conclui-se que a utilização da classificação de Russell e Norvig (1995) apresentada, e adicionalmente as sete propriedades de Franklin e Graesser (1996) enumeradas, permitem criar uma nova classificação, contendo uma classe e subclasses. Por exemplo um agente orientado a objetivo, flexível e comunicativo; dentre diversas outras combinações dentro da junção de uma classificação de Russell e Norvig seguida por um conjunto de propriedades de Franklin e Graesser.

2.2. Tecnologias

As próximas subseções apresentam as tecnologias necessárias para o desenvolvimento e interação do sistema com o ambiente. Sendo crucial o entendimento desta para a compreensão do trabalho como o todo.

2.2.1. Biometria

Talvez seja a biometria digital o exemplo mais conhecido quando se aborda o tema geral da biometria. Contudo segundo o dicionário Michaelis a biometria é definida como:

> ***1*** *Ciência da aplicação de métodos de estatística quantitativa a fatos biológicos; análise matemática de dados biológicos.* ***Var: biométrica.*** ***2 Fisiol*** *Doutrina científica e artística da medida do corpo humano.* ***3*** *Cálculo da duração provável da vida.* ***4 Espir*** *Medida das aparições espiritistas. (MICHAELIS)*

Isto posto, é possível afirmar que a biometria não se limita apenas às digitais, mas envolve também diversas outras possibilidades como por exemplo reconhecimento por voz ou reconhecimento facial.

Os sistemas biométricos são usados para a autenticação de pessoas. Nestes sistemas, existem dois modos de autenticação: a verificação e a identificação. Segundo Costa et al. (2006) a verificação se dá pela apresentação de traços biométricos do usuário juntamente com sua alegação de identidade, normalmente informada por um código de identificação. Essa abordagem de autenticação é nomeada como busca 1:1, ou busca fechada, e é feita em um BD (Banco de dados) de perfis biométricos. Assim o sistema verifica se o código de identificação informado está relacionado ao traço biométrico apresentado.

Outra abordagem de autenticação apresentada por Costa et al. (2006) é a busca aberta, ou dita uma busca 1:N. Em um BD de perfis biométricos, o sistema busca todos os registros do banco de dados e retorna uma lista de registros com características suficientemente similares à característica biométrica apresentada. É possível refinar a busca com a aplicação de outras técnicas após esse processamento ou mesmo intervenção humana.

Alguns conceitos referentes à biometria são compartilhados por todas as tecnologias biométricas. Costa et al (2006) definem um modelo simples de sistema biométrico onde são apresentados os seguintes passos:

- Aquisição e exemplar – Essa é a parte onde se realiza a leitura dos dados biométricos do usuário que devem ser balanceados conforme a necessidade da tecnologia, assim como a necessidade da precisão sem causar excessos e inconveniências para o mesmo.
- Extração e atributos – nesta etapa o exemplar é transformado em um modelo digital (*template*).
- Registro e perfil – O processo de registro faz o cadastramento efetivo no banco das informações biométricas colhidas, armazenando o *template* no banco de perfis.
- Comparação, limiar e decisão – Esse processo de comparação verifica quão similares são as características da amostra tirada e o perfil armazenado previamente. Neste processo é fornecida uma pontuação representativa dessa similaridade entre o

template e a amostra. O limiar é o quão similar essas amostras devem ser. A decisão é autenticar caso a similaridade seja maior que o limiar e rejeitar a autenticação caso contrário.

Dentre todas as tecnologias biométricas existentes, esse trabalho irá enfatizar três delas, devido a maior utilização e difusão em ambientes reais. São elas as biometrias por digital, por reconhecimento facial e por reconhecimento de voz, apresentadas a seguir.

2.2.1.1. Biometria por digital

A biometria digital é a responsável por identificar indivíduos com base no traço único presente em seus dedos. Essa tecnologia permite fazer o reconhecimento da pessoa com base na sua impressão digital.

Segundo Costa et al (2006), a biometria digital está presente desde validações utilizando papel até as validações usando sistemas de computação avançados. E seu princípio básico é verificar o nível de compatibilidade das rugosidades dos dedos entre a digital apresentada e a que procura-se verificar. Variações nas rugosidades dos dedos podem ocorrer por diversos fatores, como o passar do tempo e o clima, implicando na grande maioria desses dispositivos definirem um limiar menor que 100%.

Esse tipo de equipamento é largamente utilizado para registro de ponto. Nesse caso o equipamento tem como principal vantagem a otimização de folha de pagamento. Mas existem diversas outras aplicações para a biometria digital, como limitação de acesso a áreas restritas, controle de transferências bancárias em caixas eletrônicos sem o uso de senhas numéricas, etc.

O grau de compatibilidade para validação do usuário é definido conforme a necessidade da aplicação e velocidade desejada para os usuários serem validados pelo equipamento. Para um equipamento que registre ponto não é necessário um controle muito rígido, sendo mais interessante uma maior velocidade de validação, já que em geral equipamentos para essa aplicação têm muitos usuários para validar e normalmente em horários próximos. Por outro lado, equipamentos cuja aplicação seja a validação de um usuário para entrada em uma área com alto grau de restrição, como um cofre ou uma sala de servidor, é mais aceitável que um usuário demore um pouco mais para ser validado em prol de uma maior precisão da autenticação. Nesse caso, para maior confiabilidade, é viável considerar a utilização de outras tecnologias atuando junto a impressão digital, como senhas.

2.2.1.2. Reconhecimento facial

O reconhecimento facial pode ser utilizado em áreas de entrada restrita. Segundo Bittencourt e Osório (2000), a base dessa tecnologia é o reconhecimento de padrões em componentes de duas imagens a serem comparadas, fazendo uma classificação entre os componentes semelhantes e verificando sua compatibilidade.

Costa et al (2006) definem quatro grupos de reconhecimento facial:

a) Imagem 2D, que é subdividida em imagens estáticas e ao vivo. Ambas podem ser extraídas de qualquer câmera, entretanto a qualidade e precisão dos resultados estão diretamente ligados a qualidade do equipamento utilizado.

b) Imagem 3D, baseia-se em uma análise da geometria da cabeça, possibilitando assim que a variação da posição não interfira no resultado.

c) Sequência de imagens, se dá através da análise de imagens da face em uma sequência de imagens de câmera(s). A taxa de amostragem é geralmente mais baixa, o que dificulta seu uso em sistemas automatizados, exigindo equipamentos de custo muito elevado para obtenção de resultados satisfatórios.

d) Termograma da face, utiliza iluminação infravermelha de baixa potência. Visa resolver um dos problemas na aquisição de imagens da face que está relacionado às condições de iluminação. Essa tecnologia oferece atrativos, como a independência da iluminação ambiente e a habilidade de identificar pessoas mesmo com disfarces, mas possui alto custo de implementação. Não obstante possui como ponto negativo sofrer influência de fontes de calor.

2.2.1.3. Padrão de voz

A autenticação por padrão de voz é utilizada desde a década de 70, dividido segundo Costa et al (2006), em quatro classes:

a) **Texto Fixo:** Faz uso de uma palavra ou conjunto de palavras pré-determinadas e secretas gravadas durante a fase de registro.
b) **Dependente do Texto:** O usuário é solicitado pelo sistema de autenticação a pronunciar algo específico, dentre as diversas opções previamente registradas no sistema. Neste caso, a fase de registro é mais longa que ao do Texto Fixo. É similar ao protocolo de texto fixo, com um número maior de opções.
c) **Independente do Texto:** O usuário pronuncia frases conforme seu desejo. O sistema processa qualquer discurso do usuário.
d) **Convencional:** O usuário é interrogado, pelo sistema de autenticação, com perguntas cujas respostas são secretas, tornando-se um protocolo misto de reconhecimento e

biometria. É um protocolo similar ao dependente de texto, sendo que as frases previamente gravadas possuem um certo grau de segredo.

2.2.2. Controle de presença

Sensores de presença são largamente utilizados principalmente para controle de iluminação. Alguns pontos comerciais utilizam juntamente com um aparelho emissor de som para sinalizar a chegada de pessoas. Existem vários tipos de sensores sendo o mais popular o infravermelho.

Esse tipo de sensor é capaz de detectar a presença através da variação de calor no ambiente. Define-se através de leitura do ambiente se existe ou não alguém no local. Pode ser utilizado para ativar algum atuador ou alimentar um sistema, que por sua vez irá ativar um conjunto de atuadores específicos, conforme regras pré-estabelecidas.

2.2.3. Bancos De Dados

A evolução dos sistemas de informação trouxe um aumento de complexidade em sua construção e manutenção devido a quantidade de dados neles contidos.

Para melhor definir um banco de dados, busca-se o pensamento de Date (2003) que diz que um banco de dados é o local onde se armazenam dados necessários a atividades de uma organização no seu cotidiano com visão de uso no futuro. Ou como diz Elmasri e Navathe (2011, p. 3), “um banco de dados é uma coleção de dados relacionados, tomando a palavra dados como fatos conhecidos e que podem ser registrados e que tem significado implícito”.

Elmasri e Navathe (2011) definem banco de dados como um conjunto de dados relacionados. Para eles, a expressão Banco de Dados deixa entrever implícitas as propriedades descritas como se segue:

> *Um banco de dados é uma coleção logicamente coerente de dados com algum significado inerente. Uma variedade aleatória de dados não pode ser corretamente chamada de banco de dados.*
>
> *Um banco de dados é projetado, construído e populado com dados para uma finalidade específica. Ele possui um grupo definido de usuários e algumas aplicações previamente concebidas nas quais esses usuários estão interessados. (ELMASRI; NAVATHE, 2011, p. 3)*

Date (2004, p. 6) ajuda a confirmar a ideia dizendo que um sistema de banco de dados é “um sistema computadorizado cuja finalidade geral é armazenar informações e permitir que os usuários busquem e atualizem essas informações quando as solicitar”. Fica claro que

segundo o autor um sistema de banco de dados é composto por dados, hardware, software e usuários.

Existem diversos modelos de banco de dados. Para os intuitos deste trabalho, optou-se por priorizar o modelo relacional porque conforme apresentado por Elmasri e Navathe (2011) é um modelo simples e de base matemática, fundamentada em uma lógica de predicado de primeira ordem. O Banco de Dados Relacional (BDR) segue o modelo Relacional e sua linguagem padrão é a Structured Query Language, (SQL). Desenvolvido para facilitar o acesso a dados, o BDR possibilita aos usuários uma gama variada de abordagens no tratamento das informações.

Em banco de dados relacional há uma coleção de tabelas com um nome único. Os dados são armazenados em tabelas, que por sua vez, nada mais são do que estruturas de linhas e colunas. Uma linha em uma tabela representa um relacionamento entre um conjunto de valores. Um banco de dados pode possuir várias tabelas. Segundo Silberschatz, Korth e Sudarshan (2006) em uma tabela há uma coleção de relacionamentos com estreita relação entre o conceito de tabela e o conceito matemático de relação, a partir dos quais se origina o nome desse modelo de dados. Um exemplo de tabela de um banco de dados pode ser visto na Figura 1.1.

Nome	Cpf	Telefone_residencial	Endereco	Telefone_comercial	Idade	Media
Bruno Braga	305.610.243-51	(17)3783-1616	Rua das Paineiras, 2918	NULL	19	3,21
Carlos Kim	381.620.124-45	(17)3785-4409	Rua das Goiabeiras, 125	NULL	18	2,89
Daniel Davidson	422.111.232-70	NULL	Avenida da Paz, 3452	(17)4749-1253	25	3,53
Roberta Passos	489.220.110-08	(17)3476-9821	Rua da Consolação, 265	(17)3749-6492	28	3,93
Barbara Benson	533.690.123-80	(17)3239-8461	Rua Jardim, 7384	NULL	19	3,25

Figura 1.1: atributos de uma relação alunos Fonte: ELMASRI e NAVATHE, 2011

Silberschatz, Korth e Sudarshan (2006) lembram que esse tipo de banco de dados é útil para lidar com os problemas relacionados a aplicações administrativas e comerciais sendo a tecnologia mais difundida na área.

2.3. Ambiente Virtual

O Ambiente virtual pode ser entendido como sendo uma simulação de uma realidade e é composto por um conjunto de regras que tentam da forma mais precisa possível replicar essa realidade no computador. Essas regras visam possibilitar que, dada uma entrada de dados, seja possível gerar uma saída o mais próximo possível da que se teria em um ambiente real. Os ambientes virtuais possuem diversas aplicações que vão desde entretenimento como no caso de jogos até o seu uso em ambientes de aprendizado, como na medicina.

Tori e Kirner (2006, p. 6) definem realidade virtual como uma “interface avançada do usuário” para acessar aplicações no computador. Possuem como finalidade permitir o usuário visualizar, manipular objetos e outros elementos no cenário virtual, assim como movimentar-se nesse ambiente. Alguns desses objetos podem ser animados e possuir comportamento autômato ou disparados por eventos.

O Ambiente Virtual é a base para essa realidade virtual. A estrutura do ambiente virtual varia de acordo com a aplicação a qual este é destinado. Dessa forma alguns possuem prioridades diferentes, podendo:

> *“variar desde um ambiente com pouca fidelidade visual como no caso de reações químicas onde a representação do ambiente pode ser simples esferas coloridas representando as moléculas. Uma vez que nesse caso o mais importante é a fidelidade dos compostos apresentados e não a forma como são apresentados. Por outro lado, alguns ambientes como no caso de uma cidade onde o usuário navegará, possui a fidelidade da imagem mais importante que no caso anterior.” (Tori; Kirner,2006, p. 6)*

A virtualização de ambientes com a finalidade de estudo ou “visualização científica” como denomina Spence (2001), possui como principal necessidade criar uma visão realista do ambiente de estudo da forma mais simples e precisa possível, para que todos os aspectos cruciais sejam analisados com a menor margem de erro possível. Sendo assim, a virtualização de um mesmo ambiente possuirá aspectos diferentes de acordo com a aplicação para qual será criada. Por exemplo, a virtualização de uma casa para um jogo terá prioridades diferentes da virtualização da mesma casa no formato de uma “visualização científica” para análise de consumo de energia. Visivelmente o primeiro tem maior necessidade do aspecto gráfico e da possibilidade de interação do usuário com o ambiente e o segundo depende crucialmente da fidelidade do consumo de energia de cada equipamento ligado na casa. Sendo assim, não há perda de significância para a resolução do problema quando se apresenta a casa para o consumo de energia em um modelo 2D. Além disso, a “visualização científica” nesse caso não necessita que a geometria e a escala correspondam com a real, sendo possível que os componentes sejam identificados apenas pelo uso de formas geométricas básicas associados a legendas.

Percebe-se a importância dessa virtualização científica observando Dias e Carvalho (2007) que dissertam em seu artigo a respeito da facilidade encontrada em interpretar padrões em imagens relacionadas aos dados observados em relação a interpretações dos mesmos dados em formato de texto.

2.4. Linguagens de Programação

Os sistemas computacionais facilitam as tarefas cotidianas e, mesmo sendo inteligentes, precisam da interferência humana em sua concepção, no intuito de direcionar a máquina na realização correta das tarefas, sob pena de receber em troca respostas erradas. É neste aspecto que atua a linguagem de programação: fornecer o meio de comunicação entre computadores e humanos.

2.4.1. Definição

As linguagens de programação têm se expandido nas relações sociais e cada vez mais aproximado comunidades e informação, vida humana e facilidades cotidianas, não sendo mais restritas a grupos fechados. As linguagens dos sistemas computacionais são implementadas por algoritmos descritos de maneira que possam ser executados por um computador e percebe-se ao longo da história da computação linguagens diferenciadas de programação que cada vez mais facilitam a vida do homem.

2.4.2. Tipos de linguagem de programação

Como as linguagens humanas, as linguagens de programação são tantas quantas as necessidades evolutivas das linguagens. Schildt e Guntle (2001) ao longo de seu trabalho descrevem as diversas linguagem de programação e as divide em três níveis: alto nível, nível intermediário e baixo nível. Além disso os autores fazem uma distinção entre as linguagens em duas categorias: Linguagens não-estruturadas, e linguagens estruturadas. Estudos semelhantes são notados também em Deitel e Deitel (2011) e Stroustrup (1999).

Schildt e Guntle (2001) apresentam como exemplo de linguagem de baixo nível a linguagem Assembly, são linguagens cujas instruções correspondem quase que diretamente ao código de máquina. Os autores ainda trazem como exemplo de linguagem de nível intermediário C e C++, e para exemplificar linguagens de alto nível a Pascal e a Cobol. Adicionalmente explicam que uma linguagem de nível intermediário não significa “ser menos poderosa, difícil de usar ou menos desenvolvida que uma linguagem de alto nível” (Schildt; Guntle, 2001, p4) de forma análoga não implica possuir as dificuldades encontradas na linguagem assembly.

Dessa forma, as linguagens de nível intermediário, são a junção da liberdade de manipulação da memória do computador que o assembly possui com a facilidade de organização do código proveniente das linguagens de alto nível.

A respeito da categorização das linguagens estruturadas, Schildt e Guntle (2001) afirmam que trata-se da capacidade de compartimentalizar o código, ou seja, dividir o código em frações menores para que seja possível melhor organizar o código. Essas frações

menores são comumente descritos na literatura como funções. Assim, enquanto nas linguagens não estruturadas, um codigo é feito completamente em um bloco, nas linguagens estruturadas, podem ser divididas em funções com aplicações específicas. Essa divisão não só facilita que o código seja melhor organizado, como também facilita a reutilização de funções entre programas diferentes. Schildt e Guntle (2001) citam como linguagens estruturadas C, C++, C#, Java, dentre outras, e como não estruturadas Fortran, Basic e Cobol.

O foco deste estudo sobre linguagem de programação é o C++, um tipo de Linguagem orientadas a objetos. Segundo Schildt e Guntle (2001), o C++ originou-se a partir da necessidade de se organizar o processo de programação em uma linguagem, devido a crescente complexidade dos programas. E ela é dita linguagem orientada a objetos porque suporta o estilo de programação orientada a objetos, inspirados em outra linguagem chamada Simula67.

2.4.3. Objetivos da linguagem de programação

Define-se as linguagens orientadas a objetos como linguagens de alto nível ou nível intermediário. Diversos estudiosos apontam as linguagens de alto nível como as mais fáceis de se trabalhar e entender. Anterior à programação de uma linguagem tem-se um arquivo texto, código-fonte, onde se instrui o computador com o que se quer. Cada palavra de ordem do código-fonte corresponde à instrução. Depois de criado o código-fonte deve ser traduzido em linguagem binária usando-se para isso um compilador, que gerará um segundo arquivo executável, que é interpretado pelo computador.

É importante observar que as linguagens de programação diferem umas das outras, contendo palavras-chave próprias. A linguagem orientada a objetos pode ser de alto nível ou nível intermediário, e entre elas destaca-se a C++ que será melhor definida a seguir.

2.4.4. O C++

O C++ é uma evolução da linguagem de programação C. Bjarne Stroustrup, na década de 80, o desenvolveu incorporando à linguagem antiga as extensões suporte para a criação e uso de tipos de dados abstratos, suporte ao paradigma de programação orientada a objeto, além de diversos outras melhorias.

Schildt e Guntle (2001) dizem que C++ é um superconjunto de C, assim a maioria dos programas em C são programas em C++ também. Algumas das características de C++ são: a definição de classes, funções virtuais e *overload* de operadores para suporte à programação orientada a objetos. Assim como uso de *templates* para programação genérica, além de prover facilidades de programação de baixo nível (a exemplo do C).

Como qualquer linguagem, o C++ evoluiu demandando diversos dialetos. Contudo em 1995 os comitês de normalização da ANSI e ISO C++ chegaram a um nível de estabilidade na linguagem.

O c++ passou por um processo de padronização que demorou mais que o esperado, culminando com a versão final sendo “[...]adotada como ANSI/ISO Standard para C++ no final de 1998” (Schildt; Guntle, 2001, p440).

2.5. Heurística

Define-se Heurística como um algoritmo que não necessariamente produz o resultado ótimo do ponto de vista da otimização. A otimização toma como ótimo o resultado que seja o melhor para o problema proposto, não existindo nem um outro possível com valor melhor. Entende-se como melhor o valor mais alto existente para problemas de maximização e o menor para problemas de minimização.

2.5.1. Porque usar Heurística

Ao questionamento de por que usar um algoritmo que não garanta um resultado ótimo, Silver (2004) responde que existem pelo menos três circunstâncias que levam a essa necessidade, e são elas i) uma explosão combinatória de possíveis valores das variáveis de decisão, ii) dificuldade em avaliar a função objetivo(ou em ter restrições probabilísticas) através da presença de variáveis estocásticas, e iii) condições que mudam significativamente e uma completa série de soluções é requerida ao longo do tempo para cada item, ao invés de uma simples solução em um ponto específico no tempo.

Além disso algumas vezes na tomada de decisão o tempo disponível buscar uma resposta não é suficiente para obter ao resultado ótimo. Neste contexto, em determinados casos, “algumas vezes resultados rápidos e razoáveis são necessários e a heurística pode ser mais rapidamente desenvolvida e usada que rotinas de otimização” (Silver,2004, p8, tradução nossa).

A heurística adicionalmente é utilizada dentro de rotinas de otimização, para melhor performance na busca por uma solução. Silver (2004) afirma em seu estudo que existem três formas de fazer essa combinação. Primeiro, encontrando soluções iniciais que facilitem o trabalho das rotinas de otimização. Segundo, delimitando resultados para as rotinas, o que poupa tempo na busca, uma vez que elimina alguns conjuntos de soluções. Terceiro, a heurística é capaz de auxiliar a escolher o melhor caminho a se buscar uma resposta.

2.5.2. Metaheurística

Introduzido o conceito de heurística e sua utilidade, apresenta-se um tipo específico de heurística, a metaheurística. A metaheurística é uma subdivisão da heurística. Silver (2004) aponta a utilidade em auxiliar rotinas de otimização ou outros métodos heurísticos, para que esses obtenham soluções razoáveis para problemas matemáticos complexos no campo da otimização combinatória. Em outras palavras:

> *"A metaheurística é um conjunto de conceitos de algoritmos que podem ser usados para definir métodos aplicáveis a um vasto conjunto de problemas. O uso da metaheurística tem aumentado significativamente a habilidade de encontrar soluções de qualidade para problemas de otimização em tempos razoáveis." (Dorigo; Stützle, 2004, p25, tradução nossa)*

Nesse mesmo contexto Silver (2004) aponta como preocupações específicas da metaheurística reduzir os espaços de busca de forma sensata e não ficar presa em um local ótimo quando o problema possui mais de uma solução ótima.

Algo crucial para que a metaheurística funcione de forma ideal, é que ela possua parâmetros ajustáveis. Esses parâmetros permitem que o algoritmo se ajuste ao longo da execução, de forma a otimizar os resultados assim como o tempo levado para alcançá-los. Silver (2004) afirma que esse auto ajustes ao longo da execução permite que o algoritmo tenha uma boa flexibilidade, entretanto é imperativo que se tenha cuidado ao definir o conjunto desses parâmetros assim como seus valores máximos, mínimos e de variação.

Outro ponto interessante sobre as metaheurísticas é que algumas possibilitam um processamento paralelo, em outras palavras "a busca de diferentes sequências de soluções podem ser feitas em paralelo" (Silver 2004, p.21, tradução nossa).

Com isso conclui-se que alguns problemas possuem uma complexidade que tornam viável o uso da heurística. Além disso, em alguns casos onde a necessidade de velocidade de resposta é maior que a necessidade do resultado ótimo, a heurística deve ser considerada como abordagem de resolução do problema. Ou ainda, usar a heurística para acelerar a velocidade de execução de rotinas de otimização, seja guiando as buscas, servindo como fornecedor de respostas iniciais ou ainda provendo valores para limitar as áreas de busca da rotina de otimização.

2.5.3. Colônia de Formigas, otimização

A colônia de formigas (otimização) trata-se de um método metaheurístico proposto por Dorigo & Stützle (2004), denominado *Ant Colony Optimization* (ACO, ou em português otimização colônia de formigas). O modelo é derivado de estudos como o de Deneubourg e

outros (Deneubourg, Aron, Goss, & Pasteels, 1990; Goss et al., 1989), com o experimento da ponte dupla, que consiste basicamente em colocar um ninho de formigas ligado por duas pontes a uma fonte de alimentos, e observar o mecanismo utilizado pelas formigas para encontrar o melhor caminho dentre os dois disponíveis.

> *"Uma particular metaheurística bem sucedida é inspirada no comportamento de formigas reais. Começando com sistema de formigas, um número de abordagens de algoritmos com essa ideia foram desenvolvidos e aplicados com considerável sucesso a uma variedade de problemas de otimização combinatória tanto para aplicações acadêmicas quanto aplicações do mundo real." (Dorigo; Stützle, 2004, p25, tradução nossa)*

Dorigo & Stützle (2004), partem do modelo proposto por Deneubourg e outros (Deneubourg et al., 1990; Goss et al., 1989), que apresentam a probabilidade de uma formiga escolher um caminho baseando-se na trilha de feromônio deixada por formigas que já passaram por aquele caminho no passado. Através desse modelo foi desenvolvido o método metaheurístico citado.

O modelo se baseia na probabilidade $p_{ia}(t)$ de uma formiga no ponto de decisão $i \in \{1,2\}$ escolher um caminho $a \in \{s,l\}$onde *s* e *l* denotam o menor e o maior caminho respectivamente, no momento *t* ; *e* é dado pela função da concentração total de feromônio $\varphi_{ia}(t)$ no caminho, que é proporcional ao número de formigas que passaram por aquele caminho até o tempo *t*. O estudo apresenta a fórmula para a probabilidade de $p_{ia}(t)$ de escolher o caminho mais curto, dada por:

$$p_{is}(t) = \frac{(t_s + \varphi_{is}(t))^{\alpha}}{(t_s + \varphi_{is}(t))^{\alpha} + (t_s + \varphi_{il}(t))^{\alpha}} \quad (1.1)$$

onde: $p_{is}(t) + p_{il}(t) = 1$.

A Fórmula (1.1) foi derivada dos estudos de seguidor de caminho (Deneubourg et al.,1990) e apresentada por Dorigo & Stützle (2004) onde também afirma a similaridade do cálculo para o caminho mais longo, $p_{il}(t)$.

Dorigo & Stützle (2004) também apresentam em seu trabalho a equação diferencial para descrever a evolução do sistema estocástico como:

$$d_{\varphi_{is}}/d_t = \psi p_{js}(t - t_s) + \psi p_{is}(t), \quad (i=1,j=2;i=2,j=1), \quad (1.2)$$

$$d_{\varphi_{il}}/ d_t = \psi p_{il}(t - r \cdot t_s) + \psi p_{il}(t) \quad (i=1,j=2;i=2,j=1). \quad (1.3)$$

A equação (1.2) representa a variação instantânea no tempo *t*, do feromônio no caminho *s* no ponto de decisão *i* ; que é dada pelo fluxo de formigas ,assumindo que esse seja constante, multiplicado pela probabilidade de escolha do caminho mais curto no ponto

de decisão *j* no tempo *t* - t_s , mais o fluxo de formigas multiplicado pela probabilidade de escolha do caminho mais curto no ponto *i* no momento *t*. Com a constante t_s representando o tempo necessário para uma formiga atravessar o caminho mais curto. A equação (1.3) representa o mesmo para o caminho mais longo, com a diferença que o tempo necessário é $r \cdot t_s$. Onde *r* é a relação entre os dois caminhos, que no experimento usado para o teste foi definido como *r*= *2*.

2.5.4. Formigas Artificiais

Partindo da ideia desses estudos baseados em formigas reais, Dorigo e Stützle (2004) apresentam um modelo para ser aplicado a formigas artificiais, e visando um modelo determinístico ao contrário do apresentado nas Fórmulas (1.1) a (1.3) que representam um cenário contínuo. Assim esse modelo determinístico assume que a cada passo do tempo as formigas se movem para um nó adjacente a uma velocidade constante (uma unidade de distância por unidade de tempo), além disso elas adicionam uma unidade de feromônio para o caminho que elas utilizam.

Nesse modelo apresentado as formigas se movem no grafo (pelos caminhos de um nó para outro) utilizando de um método de escolha probabilístico: $p_{is}(t)$ que é a probabilidade de uma formiga localizada no nó *i* no tempo *t* escolher o caminho *s*, e de forma equivalente $p_{il}(t)$ que é a probabilidade de para o caminho *l*. Essas probabilidades se baseiam na função de feromônio no caminho φ_{ia} que as formigas no nó $i\,(i \in \{1,2\})$encontre no caminho $a, (a \in \{s,l\})$:

$$p_{is}(t) = \frac{[\varphi_{is}(t)]^{\alpha}}{[\varphi_{is}(t)]^{\alpha} + [\varphi_{il}(t)]^{\alpha}}, \quad p_{il}(t) = \frac{[\varphi_{il}(t)]^{\alpha}}{[\varphi_{is}(t)]^{\alpha} + [\varphi_{il}(t)]^{\alpha}}. \tag{1.4}$$

Onde a atualização da trilha de feromônios no caminho é dado por:

$$\varphi_{is}(t) = \varphi_{is}(t\text{-}1) + p_{is}(t\text{-}1)m_i(t\text{-}1) + p_{js}(t\text{-}1)m_j(t\text{-}1),$$
$$(i=1,j=2;i=2,j=1), \tag{1.5}$$

$$\varphi_{il}(t) = \varphi_{il}(t\text{-}1) + p_{il}(t\text{-}1)m_i(t\text{-}1) + p_{jl}(t\text{-}r)m_j(t\text{-}r),$$
$$(i=1,j=2;i=2,j=1), \tag{1.6}$$

sendo $m_i(t)$ o número de formigas no nó *i* no instante *j*, que é representado por:

$$m_i(t) = p_{js}(t\text{-}1)m_j(t\text{-}1) + p_{jl}(t\text{-}r)m_j(t\text{-}r), (i=1,j=2;i=2,j=1), \tag{1.7}$$

Esse modelo proposto para as formigas artificiais difere do modelo proposto na Seção 2.5.3 em dois importantes aspectos, segundo Dorigo e Stützle (2004):

- considera o comportamento médio do sistema, e não apenas o comportamento estocástico de uma simples formiga.
- é um modelo discreto (referente ao tempo), enquanto o anterior é um modelo de tempo contínuo; portanto usa diferenças no lugar de equações diferenciais.

Afirma-se embasado nos autores citados nessa Seção, que alguns problemas são melhores tratados com o uso da heurística. Toma-se os modelos apresentados nas subseções 2.5.3 e 2.5.4 como caso de sucesso, sendo a proposta desses modelos a resolução do problema de caminho mínimo, testado com sucesso para o caso do problema da ponte dupla apresentado no estudo de Dorigo e Stüzle (2004).

Conclui-se assim que uma vez que a heurística permite ser combinada com outras heurísticas e/ou com rotinas de otimização, tem-se nessa técnica uma ferramenta eficiente para tratar diversos tipos de problemas, alguns dos quais são inviáveis de serem tratados apenas com rotinas de otimização.

2.6. Trabalhos Relacionados

Os primeiros passos no sentido de criar-se casas inteligentes foi a produção de automações residenciais, as quais foram de suma importância para iniciar o processo da domótica.

Prudente (2015) apresenta três anexos com projetos de automação residencial, com foco na resolução de problemas isolados relativos a automação residencial e predial. Essas soluções são apresentadas de forma prática com o intuito de servir de guia para a implementação dos mesmos.

O primeiro anexo apresenta o projeto domótico de instalação de persianas motorizadas usando o sistema My Home. No sistema, os motores monofásicos de acionamento das persianas são comandados por atuadores embutidos posicionados perto da janela, em que cada comando dado faz subir ou descer a persiana.

O segundo apresenta o projeto domótico completo de instalação de uma sala de sala de reunião em um edifício, usando o sistema SCS de automação que permite a gestão de iluminação e dos equipamentos multimídias da sala. Nesta sala, os dispositivos de comando ficam em uma pequena central de controle, de onde se controla todo o ambiente.

O último anexo apresenta o projeto domótico de instalação do sistema de gestão de energia para controle de cozinha e banheiro residenciais. Neste caso o sistema é configurado de forma que haja intervenção em caso de sobrecargas.

Messias (2007) propõe um *check-list* visando facilitar uma implementação de sistemas domóticos, seja em residências, edifícios ou micro e pequenas empresas com soluções de automação de baixo custo. O enfoque do trabalho por ele apresentado está na combinação de soluções já existentes agregadas em pontos de controle, tais como iluminação, segurança e Telecomunicações. O resultado do trabalho foi um guia básico com os principais pontos de controle que devem ser implementados em sistema domótico.

McGlinn et al. (2010) propõem uma solução de baixo custo para implementação de sistemas de construções inteligentes ou *smart buildings* (SB) através de simulações em ambientes virtuais, evitando assim custos de hardware para testar a viabilidade e a funcionalidade das SB. Essa proposta de trabalho é apresentada com a justificativa que os desenvolvedores idealmente precisam testar seus protótipos cedo e de forma repetida com o menor custo possível. O que é proporcionado através da virtualização e simulação do ambiente, como proposto em seu trabalho.

McGlinn et al. (2010) apresentam uma discussão sobre ferramentas que dão suporte à modelagem e simulação de SB's para avaliar a dependência de contexto para *Smart Buildings Aplications* (SBA) para em seguida dissertar sobre a construção de ferramentas que possibilitam a visualização e simulação de SB´s para suporte em processos de validação. Ao final do trabalho é proposta uma plataforma com a finalidade de dar suporte a desenvolvedores de SBA's através da simulação dos SB's e suas respostas ao SBA, possibilitando um desenvolvimento mais preciso dessas aplicações assim como a validação do mesmo.

3. METODOLOGIA

Esse Capítulo apresenta a modelagem da solução do problema apresentado neste trabalho, assim como as técnicas utilizadas em cada etapa para o cumprimento dos objetivos propostos.

3.1. Formulação do Problema

O princípio básico da domótica é aumentar o conforto e a segurança das pessoas por meio de tecnologias de automação implantadas em suas residências. Viu-se também os possíveis benefícios obtidos através de um agente inteligente integrado a essa tecnologia. Dessa forma torna-se importante uma tecnologia que possibilite a integração e controle de todos esses equipamentos e faça uso da A.I.

A formulação do problema será dividida em dois pontos, o sistema unificado de controle domótico, doravante denominado *Framework* AIHC e o módulo de inteligência artificial a ser integrado a esse sistema.

3.1.1. Formalização do *Framework* AIHC

Para que o *Framework* atenda às necessidades de um sistema domótico, existe um conjunto mínimo de tecnologias que devem ser controladas, tais como sensores e equipamentos, dando ênfase a sensores biométricos e de detecção de movimento. Dentre os equipamentos eletroeletrônicos é possível citar as portas digitais, controle de lâmpadas, ar-condicionado e alarmes.

Esse *Framework* deve ser construído de forma modular, o que permite uma maior manutenibilidade. O fato de ser um sistema modular garante a possibilidade de adição de novos módulos de acordo com a necessidade de cada usuário. Além disso permite que modificações em um módulo não comprometam o funcionamento de outro.

O controle de aparelhos simples já existe em larga escala. São problemas triviais que podem ser resolvidos com o uso de tecnologias do tipo Arduino, que, além de possuir baixo custo, é um hardware livre que possui inúmeros módulos compatíveis. Entretanto esse controle deve ser incluído no escopo de uma plataforma única.

Avanços tecnológicos significativos em relação a transcrição de texto para voz e vice-versa trazem consigo uma nova tendência para o mercado a interação multimodal. Caso essa tendência se consolide e torne-se uma exigência do mercado, é interessante que uma tecnologia de controle residencial possua esse conceito implementado, mesmo que

inicialmente apenas no formato de texto, permitindo modificações futuras para implementar comandos de voz.

Assim sumariza-se o *Framework* AIHC como um sistema modular, composto por dois módulos principais: i) AIHC responsável por fazer o controle de leituras dos sensores as comparando com as regras de automação e criando comandos quando necessário, ii) e o Controle, responsável por manter os equipamentos nos estados devidos. Além disso, 3 drive's dão suporte ao funcionamento do *Framework*, sendo: i) Sensor, responsável por captar as leituras e registra-las no banco de dados, ii) Equipamento, responsável pela mudança de estado dos equipamentos cadastrados no *Framework* e iii) Ouvido Externo, que atua como uma interface com usuário, com suporte a comando de textos. Essas relações podem ser vistas na Figura 2.1.

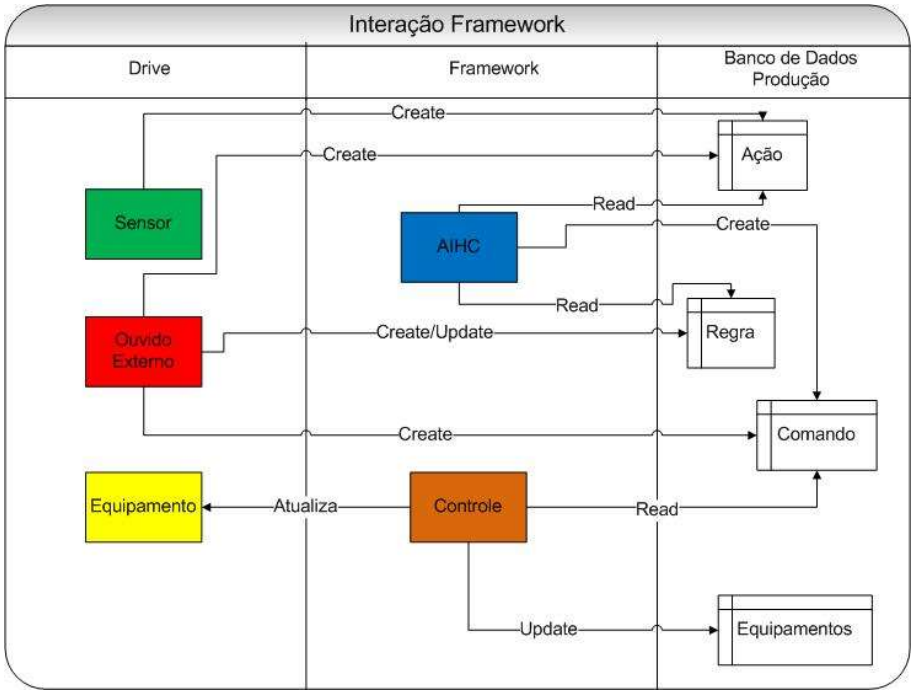


Figura 2.1: Visão *Framework* AIHC Fonte: Elaborado pelo autor

3.1.2. Formalização da Inteligência Artificial

A importância da inteligência artificial no presente trabalho está relacionada a facilidade que ela traz para os usuários. Através da análise de recorrências em relação ao uso dos equipamentos e leituras dos sensores, o sistema deve produzir regras de controle para os equipamentos e em seguida perguntar ao usuário se essa regra realmente deve ser implementada. Essa interação com o usuário auxilia nas decisões de regras de otimização para maior comodidade e segurança da residência e seus moradores.

O ambiente ao qual o agente será inserido é uma casa contendo sensores, atuadores e a presença de outros agentes como, por exemplo, as pessoas. O agente terá acesso a todas as leituras dos sensores e os estados dos equipamentos, através das informações contidas no Banco de dados Central (CDB).

De acordo com a classificação proposta por Russel e Norvig (1995) para uma definição formal do ambiente, pode-se classificar o entorno da casa como acessível, pois o agente inteligente terá acesso a todas as informações do ambiente a todo momento. Além disso, é um ambiente não determinístico, uma vez que as pessoas dentro da casa eventualmente mudam o estado do ambiente, tal que o controle não cabe exclusivamente ao agente inteligente.

Em relação a episodicidade do ambiente, o ambiente é classificado como não episódico, onde as ações tomadas pelo agente irão influenciar na tomada de decisões futuras.

Nitidamente o ambiente de atuação do agente será dinâmico, uma vez que as pessoas não irão ficar paradas à espera do agente tomar uma decisão. E por fim, o agente irá atuar em um ambiente contínuo, uma vez que a existência de pessoas impossibilita que o agente preveja todas as possíveis mudanças de estado.

Uma vez que o ambiente foi formalmente classificado, é possível agora formalizar a definição do agente que será criado para resolução do problema. A classificação para o agente seguirá o modelo proposto por Russell e Norvig (1995), adicionando as propriedades sugeridas por Franklin e Graesser (1996), definindo assim a classe e subclasse do agente.

O objetivo principal é a criação de regras para aprovação do usuário. Assim, o agente poderia ser classificado como Agente Orientado a Objetivo. Entretanto, um agente construído apenas observando esse ponto apresentaria um de dois problemas: a) demasiadas propostas de regras, o que seria resolvido com a adição de outros objetivos, mas que teriam como consequência outro problema: b) uma estagnação de propostas dentro de um período relativamente curto, não garantindo encontrar as regras de otimização que contribuam para economia e comodidade, justamente o problema a solucionar. Conclui-se então que o ideal é optar por um Agente Orientado a Utilidade, tendo o objetivo de propor regras, mas como uma avaliação da utilidade de cada regra a ser proposta, evitando sugerir regras desnecessárias sem pena de estagnar-se.

Com a classificação principal do agente, cabe agora a definição da subclasse, averiguando as propriedades importantes para seu melhor funcionamento. Devido ao impacto psicológico e social que teria uma casa completamente autônoma, esse atributo não será utilizado para construir o agente aqui proposto. É necessário que ele seja comunicativo, recebendo entradas de humanos como forma de controle. É imprescindível que sua execução ocorra pelo tempo em que a casa existir, assim é crucial que seja um agente de tempo

contínuo. E por fim é primordial que aprenda com suas experiências, como já definido na classificação do ambiente, o que justifica a inclusão do aprendizado.

Dado o anterior, definiu-se a quantidade de variáveis a serem tratadas pelo problema, como pode ser visto na Fórmula 2.1

$$PR = (NS * NDL) * (NE * NDE) \quad (2.1)$$

A interpretação da Fórmula é a que se segue: Possíveis regras (PR) é igual ao número de sensores (NS) multiplicado pela média de possíveis leituras (NDL); multiplicado pelo número de equipamentos (NE) multiplicado pela média de possíveis estados (NDE).

Seguindo a Fórmula 2.1 e tomando por hipótese uma casa de 6 cômodos, com dois sensores em cada cômodo, supondo ainda que esses sensores terão apenas leituras binárias (2 possibilidades de leituras), e ainda que nessa casa haja 3 equipamentos em cada cômodo, e que a média de estados desses seja 3, tem-se que PR = ((2*6)*(2)) * ((3*6) * 3). Logo, PR = 1296.

Nota-se que parte do problema é o grande número de possibilidades de relações que podem surgir em um sistema domótico complexo. Ressaltando a necessidade de que um agente inteligente seja capaz de reconhecer a relevância de cada relação, espera-se que o agente se limite a sugerir as regras com maior potencial de aceitação, aprendendo com suas experiências e aprimorando sua percepção de relevância.

Supõe-se também que o usuário terá um conjunto X de regras aceitáveis que a A.I. deve deduzir. Esse conjunto X é um subconjunto de Y(o conjunto de todas as possíveis regras). Assim, usando a hipótese citada onde PR = 1296, o conjunto Y possui 2^{1296} subconjuntos, uma vez que esse é o número de subconjuntos possíveis para um conjunto de 1296 elementos. Logo existem 2^{1296} possibilidades de resposta.

Objetiva-se que a A.I. encontre o conjunto X, minimizando perguntas ao usuário, uma vez que um grande volume de perguntas venha a ser um inconveniente, mas também deve descobrir o mais rápido possível qual o conjunto X. Para isso é preciso que suas métricas e parâmetros de controle de qualidade sejam recalculados durante o processo de aprendizagem e conforme ressaltado por Silver (2004) a calibragem dessas métricas e parâmetros deve ser meticulosa.

Ressalta-se que o usuário pode, a qualquer momento, mudar de ideia a respeito de determinada regra. É importante incluir essa variação de opinião do usuário no escopo da A.I.

Sumariza-se a formalização do agente inteligente como sendo um agente inteligente inserido em um ambiente classificado como: **acessível, não determinístico, não episódico,**

dinâmico e contínuo. E em relação a classificação, o mesmo pode ser considerado um: ***agente inteligente orientado a utilidade, contínuo, comunicativo e com aprendizagem.*** Por fim, o perfil estocástico do problema, conforme Fórmula 2.1, justifica o uso de uma **metaheurística** para obtenção de resultados satisfatórios em um tempo de processamento aceitável.

3.2. Análise de Requisitos

Na Seção anterior foi possível verificar a definição formal do *Framework* AIHC, do ambiente e do agente objetos do presente trabalho. Agora será feita uma análise das funcionalidades que esta plataforma deve conter. Os quatro elementos principais que viabilizam o sistema são:

a) O *Framework*, que será responsável por controlar os equipamentos e sensores, tendo a responsabilidade de executar as regras específicas sempre que as condições definidas nas regras sejam atingidas, atuando como a maioria dos sistemas domóticos,

b) O CDB, cujo objetivo é armazenar em suas tabelas dados importantes. Alguns desses dados são: as regras adicionadas pelo usuário (Regras Específicas), os cadastros de pessoas (Pessoa), cômodos da residência (Cômodo), equipamentos instalados (Equipamento), sensores instalados (Sensor) e atuadores presentes na residência (Atuador). Deve também registrar todas as leituras de sensores feitas e todas as mudanças de estados dos equipamentos;

c) O módulo de inteligência artificial, que será responsável por acompanhar pelo CDB as leituras dos sensores e os comandos enviados para os equipamentos buscando definir recorrências que porventura possam se tornar regras de automação, seja para aumentar a comodidade ou otimizar o uso de energia.

d) O Banco de Dados de Conhecimento (KDB), que será responsável por armazenar os dados filtrados pelo agente inteligente, tais como: registros de recorrência, as regras negadas, as possíveis regras a serem propostas e as métricas de avaliação de desempenho do sistema. Os parâmetros para definir se uma regra é viável ou não, também serão armazenados no KDB. Esses dois últimos (métricas e parâmetros) terão seus valores reajustados à medida que o sistema interage com o usuário. Aumentando ou diminuindo seus valores conforme vai “aprendendo” dependendo das respostas para cada regra proposta. Sumarizando, o KDB é a base de conhecimento do agente inteligente, e representa seu aprendizado.

Para melhor entendimento dessa relação toma-se por base a visualização da Figura 2.2, na qual percebe-se as relações entre os principais módulos de forma geral.

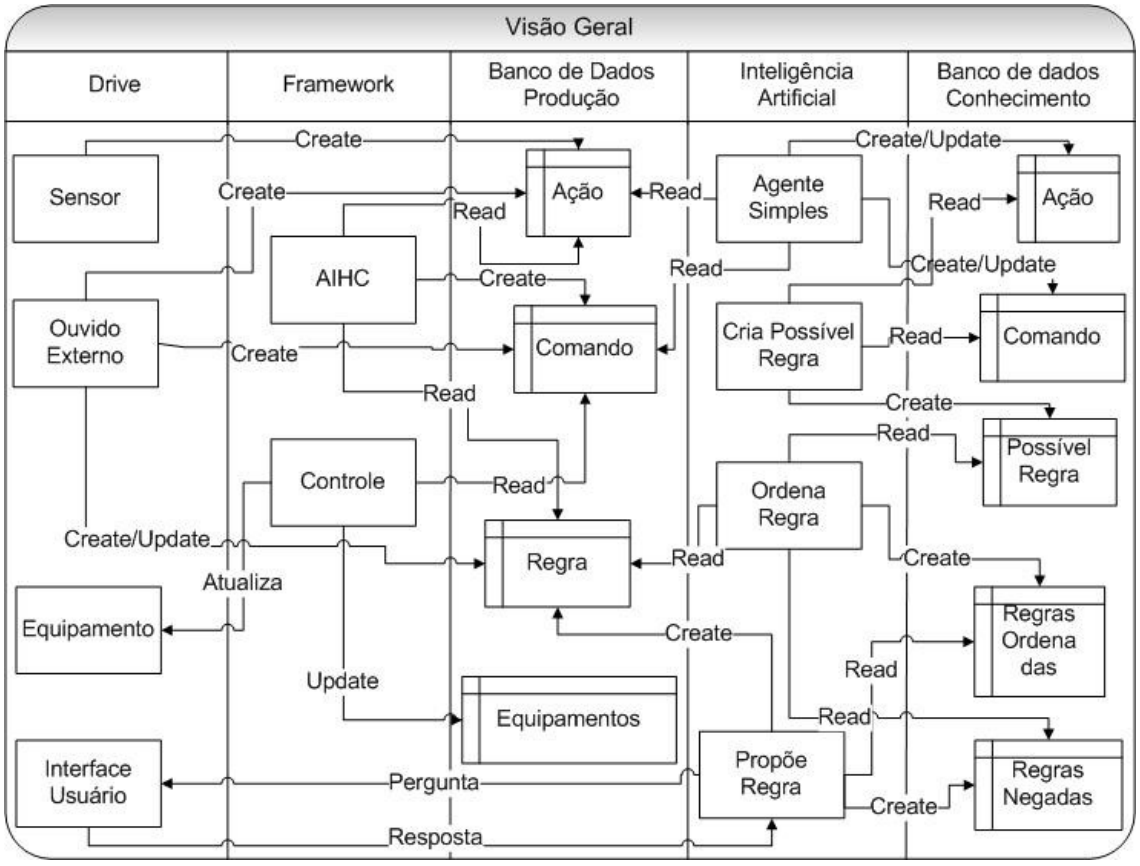


Figura 2.2: Visão Geral Fonte: Elaborado pelo autor

Através da Figura 2.2 observa-se um conjunto de quatro *drive's* que se conectam ao sistema.

- Um de Sensor, que fornece as entradas de sensores, registrando-as no CDB;
- um Ouvido Externo, que é a interface de comunicação com *Framework* através de registros armazenados no CDB;
- um de Equipamentos, que recebe as alterações do Controle do *Framework*, mantendo os estados dos equipamentos atualizados;
- e um Interface Usuário, que faz a comunicação do usuário com a Inteligência Artificial.

Nota-se a existência de dois componentes no *Framework*, o *Kernel* e o Controle. O primeiro se encarrega de acompanhar o CDB comparando as leituras de sensores com as regras e inserindo no CDB comandos a partir dessa comparação (quando uma leitura dispara uma regra, ou seja, deve ser feita uma mudança de estado de equipamento). O segundo, se encarrega de acompanhar no CDB todos os comandos registrados, sejam provenientes do

Kernel ou do Ouvido Externo, enviando alteração de estado para o drive equipamento, assim como atualizar o estado do equipamento no CDB.

O CDB, armazena informações sobre as ações (leituras de sensores), comandos (para mudança de estado dos equipamentos), as regras (de automação, criadas pela Inteligência Artificial ou diretamente pelo usuário), e estado de equipamentos, além dos dados referente a cadastros do *Framewok*.

A inteligência artificial possui quatro componentes principais:

a) um agente simples, que se encarrega de fazer a leitura das ações e comandos no CDB filtrando os mais relevantes e armazenando-as no KDB;

b)um Cria Possivel Regras, que avalia a compatibilidade da ação e comando, associando-os e os armazenando em Possível Regra no KDB;

c) um Ordena Regra, que avalia as possíveis regras de acordo com o índice de aceitação e rejeição de regras similares, e as salva em uma ordenação decrescente no Regras Ordenadas do KDB;

d) e um Propõe Regras, que através da seleção das regras ordenadas apresenta através da Interface Usuário as regras para o usuário, armazenando as aceitas em Regra do CDB e as rejeitadas em Regras Negadas do KDB.

Por fim o Banco de dados Conhecimento se encarrega de armazenar os dados de ação e comando relevantes a inteligência artificial, as possíveis regras derivadas por esse, a ordenação dessas possíveis regras e as regras que foram negadas. Além de dados para o controle de qualidade do agente inteligente.

3.3. Modelo

A modelagem da plataforma visa a construção de um software modularizado, aumentando sua manutenibilidade. Seguindo essa linha de raciocínio definiu-se os módulos listados a seguir como o padrão a ser adotado. A modelagem do sistema é dividida seguindo os itens propostos como requisitos do sistema apresentados na Seção anterior.

Destarte as próximas subseções irão explanar a modelagem do *Framework* AIHC, a construção dos bancos de dados (tanto o CDB quanto o KDB), a modelagem do Agente Inteligente e por fim a modelagem do ambiente de simulação.

3.3.1. *Framework* AIHC

O *Framework* está subdividido em quatro módulos conforme suas funções, de acordo com a Figura 2.3. O objetivo dessa divisão é manter o sistema o mais modular possível, facilitando sua manutenção e reutilização.

FRAMEWORK AIHC			
1	**Cadastros**	**2**	**Kernel**
1.1	Cadastro	2.1	Controle de sensores
1.2	Alteração	2.2	Leitura de Regras
1.3	Remoção	2.3	Criação de comandos
1.4	Integração	2.4	Integração
3	**Controle**	**4**	**Interface Com Usuário**
3.1	Controle Efetivo	4.1	Regras De Interação
3.2	Integração	4.2	Integração

Figura 2.3: Divisão do *Framework* Fonte: Elaborado pelo autor

No módulo de cadastro, todas as funções de cadastro são semelhantes entre si, e objetivam cadastrar equipamentos, sensores, pessoas, atuadores e portas. De forma equivalente aos cadastros são as alterações e remoções.

No *Kernel*, o controle de sensores acompanha as leituras registradas; o componente “Leitura de regras”, recebe as leituras e as compara com as regras existentes. Caso exista uma ou mais regras para a leitura feita, envia a regra para criação de comando, que cria comando(s) de acordo com o especificado na(s) regra(s).

No módulo Controle, a função “controle efetivo” é utilizada para enviar comandos aos equipamentos cadastrados, sejam provenientes do *Kernel* ou de um comando direto do usuário. Esse módulo consiste nos comandos de ativação e desativação dos equipamentos. Também é o responsável pela alteração de estado para equipamentos como ar-condicionado por exemplo, que além do estado desligado/ligado possui uma variável temperatura.

A função “Integração” é a responsável por integrar o módulo ao *Framework*, disponibilizando uma padronização de comunicação entre o *Framework* e o módulo em questão. Em outras palavras essa é a funcionalidade que fornece dados para o CDB, possibilitando a integração da informação.

No módulo Interface com Usuário a funcionalidade “regras de interação” é responsável pela comunicação do usuário com o *Framework*, através do envio de comandos e criação/alteração de regras, assim como a realização de cadastros; e com a inteligência artificial (o Agente Inteligente) pelo envio de perguntas para que o usuário responda o sistema, tal como o retorno da resposta. Faz também a conexão da interface com o usuário e sensores de presença e biométricos para garantir a autenticidade das respostas.

3.3.2. Banco de dados

O sistema AIHC possui dois bancos de dados distintos, o CDB (AIHC.sql) que registra o ambiente real e o KDB (knowledge.sql) que armazena os dados trabalhados pelo Agente

Inteligente, assim como as métricas e parâmetros que serão apresentados adiante. Essa separação objetiva que o sistema de produção não seja comprometido pelas buscas de relações da A.I. no banco, além de permitir uma melhor organização do sistema.

O KDB possui duas funções conceituais básicas. Servir de base para armazenar as ações e comandos que refletem uma recorrência que seja viável tornar-se uma sugestão de regra específica, e armazenar as sugestões de união entre essas para que sejam avaliadas pelos filtros do sistema, e apresentadas para o M.U.

Desta forma a utilização de banco de dados relacional é suficiente para prover todos os serviços de banco de dados necessários para essa plataforma dos conjuntos básicos. A Figura 2.4 apresenta às tabelas do CDB e as ligações entre elas, o banco completo pode ser visto no Anexo I.

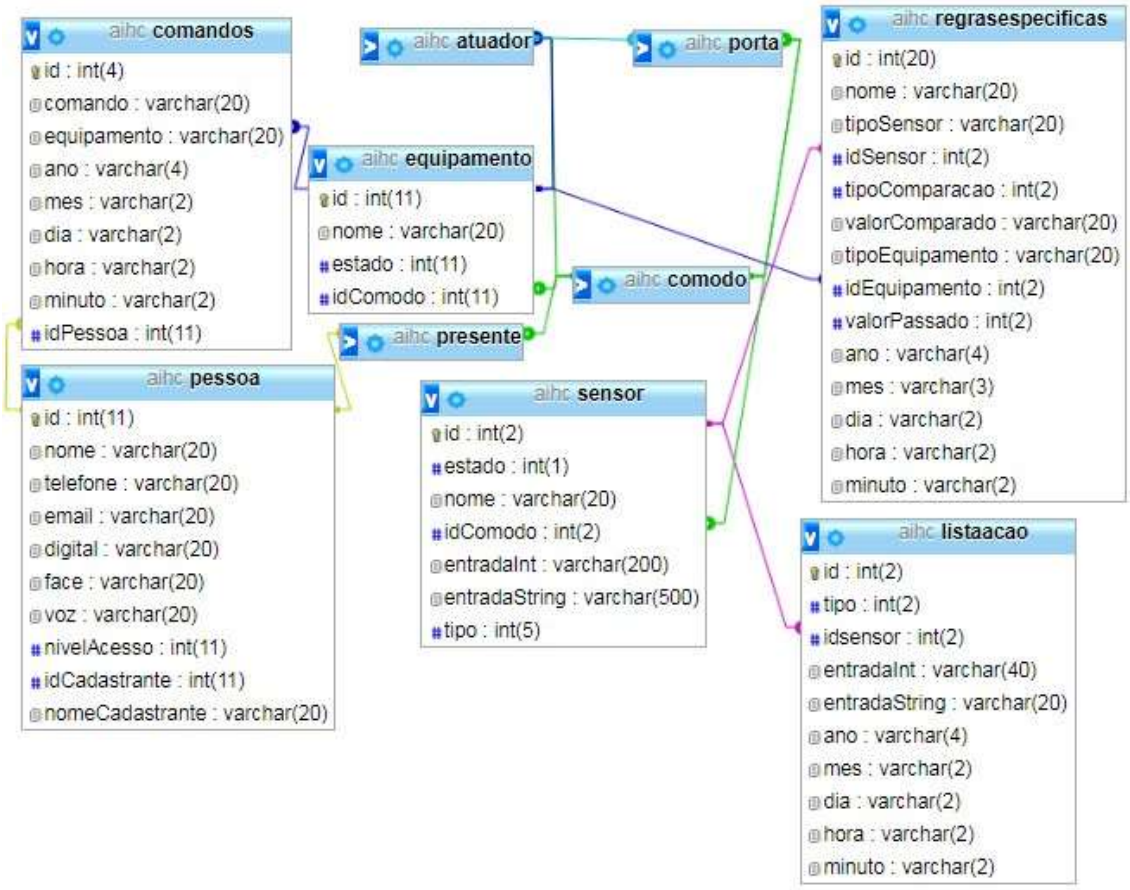


Figura 2.4: Banco de Dados Central Fonte: Elaborado pelo autor

A Figura 2.5 apresenta as tabelas do KDB, o qual armazena os dados necessários para que o módulo de inteligência artificial gere seu conhecimento, através da análise dos dados armazenados nesse banco, interação com o usuário e a inteligência artificial (que também são armazenadas nesse banco) e observação do ambiente real (também modelado

para atender essa demanda e após isso armazenado no banco de conhecimento). O banco completo pode ser visto no Anexo I.

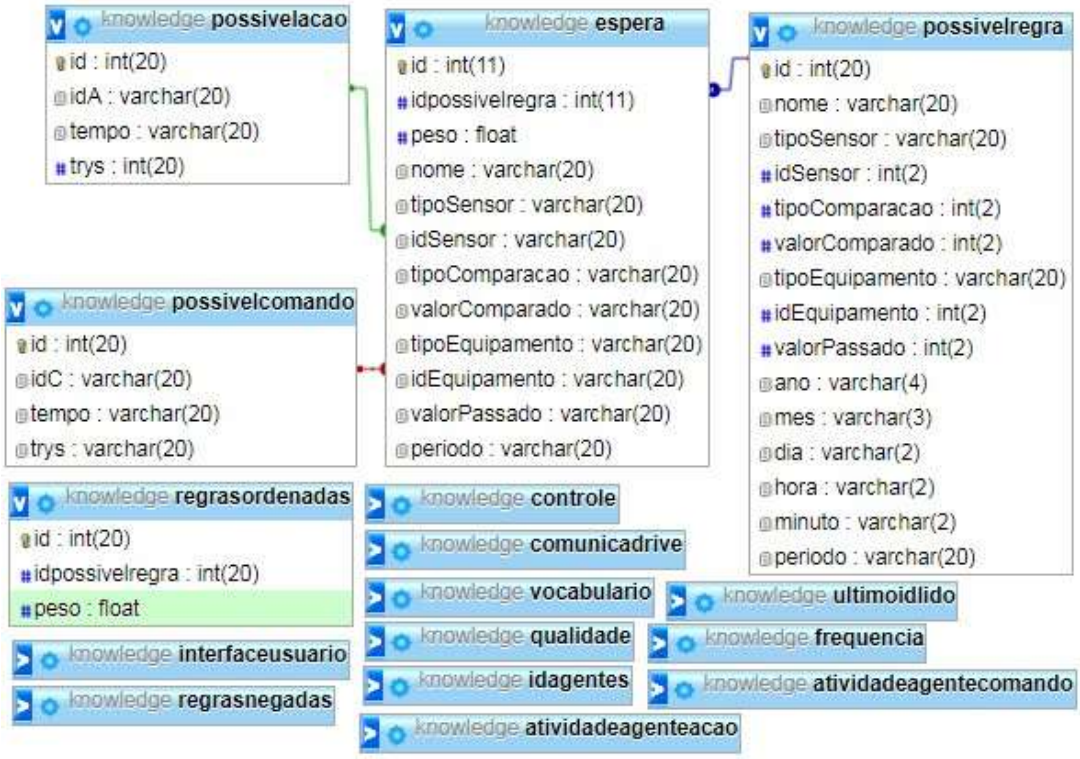


Figura 2.5: Banco de Dados Conhecimento Fonte: Elaborado pelo autor

O projeto do banco inclui a construção de um CRUD (Create, Read, Update e Delete) genérico, para fazer a conexão entre as tabelas e código em C++. O CRUD permite criar novos dados (Create), recuperar dados das tabelas (Read), alterar campos (Update) e apagar registros das tabelas (Delete).

3.3.3. Agente Inteligente

O Módulo “Agente Inteligente” é o módulo responsável pela A.I. do sistema. Ele é subdividido em 5 módulos menores, que efetuam cada um uma parte do processo de geração de regras. É também o responsável pelo controle dos parâmetros do sistema, utilizados para avaliar os dados originados do *Framework*. Esses cinco módulos são, Agente Simples, Gerador de Regras, Ordenador de Regras, Proponente de Regras e Qualidade Total, e serão apresentados nesta subseção seguindo essa mesma ordem. As variáveis das Formulas desta Seção podem ser vistas na Tabela do Anexo II.

O **Agente Simples** é o responsável por fazer a varredura nas tabelas de ações e comandos. Comparando as recorrências *R* no período *t* de leituras *l* do sensor *s* e as recorrências no período de valores *v* de equipamentos *e*, com o valor de frequência mínima μ no momento *a*, salvo no banco de conhecimento para o período *t*. Caso a recorrência atenda os padrões definidos na tabela de frequência, cria-se um registro *G* desse comando ou ação em uma tabela reservada para isso no banco de conhecimento referente ao período *t*. Essa rotina é expressa como:

$$G_t(s,l) = R_t(s,l) > \mu_{at} \quad G \in \{true,false\} \tag{3.1}$$

$$G_t(e,v) = R_t(e,v) > \mu_{at} \quad G \in \{true,false\} \tag{3.2}$$

Onde:

$t = [t_0,t_a]$ || t_0 é o início do período e t_a é o momento atual (3.3)

As Fórmulas (3.1), (3.2) e (3.3) são: a gravação *G* de uma tupla *x* para o período *t*, depende da recorrência *R* no período *t* dessa mesma tupla ser maior que o μ determinado no momento atual para o período *t*. Onde: $x \in \{(s,l),(e,v)\}$. Logo,

$$\forall \ G_t(x) = true, \exists S_t(x) \ , \forall \ x \in \{(s,l),(e,v)\}. \tag{3.4}$$

Com:

S representando um registro salvo no banco referente ao período *t*.

O **Gerador de Regras** é o módulo que tem como finalidade buscar na tabela *S* supracitada e comparar o período de cada recorrência para determinar se a ação *(s,l)* e o comando *(e,v)* possuem uma relação *A* entre eles. A relação *A* é expressa por:

$$A_t(i_{sl},i_{ev}) = true, se\ (\exists z,b)\ (\ z = b) \mid z \in H_t(i_{sl}), b \in H_t(i_{ev}), \tag{3.5}$$

$$A_t(i_{sl},i_{ev}) = false, se\ (\forall z,b)(\ z \neq b) \mid z \in H_t(i_{sl}), b \in H_t(i_{ev}), \tag{3.6}$$

Onde:

i_{xy} representa o id da tupla *(x,y)* na tabela *S* com $x \in \{s,e\}$ e $y \in \{l,v\}$ enquanto $H_t(i_{xy})$ representa o conjunto de todas as horas das recorrências das tuplas *(x,y)* no período *t*.

Caso exista a relação $A_t(i_{sl},i_{ev}) = true$, esse módulo consulta se já existe uma regra específica Ω correspondente à tupla (i_{sl},i_{ev}).

Caso não exista $\Omega(i_{sl},i_{ev})$, consulta se existe uma ou mais recorrência que corresponda a essa tupla na tabela de regras negadas Θ, e o módulo carrega a fórmula de viabilidade *V* para o período *t* segundo o valor de $\Theta(i_{sl},i_{ev})$, que representa a quantidade de vezes que essa regra foi negada pelo(s) usuário(s). A qual é expressa pela seguinte Fórmula:

$$V_t(i_{sl},i_{ev}) = \neg(\exists\Omega(i_{sl},i_{ev})) \wedge\{ (\Theta(i_{sl},i_{ev}) \cdot\mu_{at}\cdot\kappa) <\delta_t(i_{sl},i_{ev})\} \quad (3.7)$$

Onde:

$$\delta_t(i_{sl},i_{ev}) = R_t(e,v), \text{ se } R_t(e,v) \leq R_t(s,l) \quad (3.8)$$

$$\delta_t(i_{sl},i_{ev}) = R_t(s,l), \text{ se } R_t(e,v) > R_t(s,l) \quad (3.9)$$

Onde:

μ_{at} é o valor definido para a frequência mínima requerida no momento atual a para o período definido *t*

κ é uma variável parametrizável pela inteligência artificial, cuja função é limitar o excesso de aparições de regras repetidas.

Desta maneira, uma vez que a possível regra ainda não esteja na tabela de regras específicas e atenda ao valor mínimo da fórmula de viabilidade é criado um registro na tabela de possível regra P, para avaliação do módulo ordenador de regras.

O **Ordenador de Regras** é o módulo responsável por ponderar cada possível regra P, avaliando quais são as regras mais relevantes. Essa ordenação é feita sempre que uma determinada cota c de regras P é alcançada. Assim, essa ordenação ocorre sempre que:

$$P_a = P_b + c \quad (3.10)$$

Onde: P_a é o total de possível regra em P no momento atual e P_b é o total de regras em P no momento da última vez que as regras foram ordenadas.

Essa cota c é uma variável usada como parâmetro de controle. Seu valor é alterado no módulo de qualidade apresentado adiante.

Após *P* alcançar o valor necessário para satisfazer a condição da cota, é feita a ordenação das regras no intervalo $]P_b,P_a]$ através do peso associado *VT* de cada *P* ,que é dado por:

$$VT_{i_s i_e} = ((\rho_e\cdot 2) +(\rho_s \cdot 3) + (\rho_{se} \cdot 5)) \cdot\gamma_{se} \quad (3.11)$$

A Fórmula (3.11) é lida como: o peso *VT* para uma combinação do sensor ***s*** com o equipamento ***e***, é igual a duas vezes a probabilidade ρ de aceitar o equipamento ***e***, mais três vezes a probabilidade ρ de aceitar o sensor ***s***, mais cinco vezes a probabilidade ρ de aceitar o sensor ***s*** com o equipamento ***e***, essa soma multiplicada pelo peso γ das distâncias

geográficas do sensor *s* para o equipamento *e*. As constantes utilizadas foram definidas empiricamente, e sua justificativa é apresentada na conclusão da Fórmula (3.11) adiante. Com essa Fórmula afirma-se que:

$$\forall x, VT_x \in GP \mid GP = [0,10],\ GP \in R,\ x \in P_x \tag{3.12}$$

Ou seja, para todo *x* o valor de VT_x é um número real que pertence ao intervalo fechado de zero a 10, com *x* pertencendo ao conjunto de regras *P* com a tupla representada por *x* desconsiderando-se os valores de leitura *l* do sensor assim como o valor de estado *v* do equipamento.

A probabilidade ρ apresentada na equação (3.11) foi baseada na heurística ACO proposta no trabalho de Dorigo & Strüdzle (2004) apresentado nas seções 2.5.3 e 2.5.4. O problema proposto por eles consiste na escolha entre dois caminhos, onde o caminho com maior acúmulo de feromônio tende a ser o melhor.

O uso principal dessa heurística é encontrar o menor caminho em um grafo. De forma simplificada, o problema aqui proposto é avaliar a probabilidade do usuário aceitar uma regra proposta. Sumarizando, dado uma proposta *X*, existem duas possibilidades i) o usuário aceitar *X*, ou ii) o usuário rejeitar *X*. Afirma-se então que existem dois caminhos a partir de *X*, i e ii que remetem as duas possibilidades descritas respectivamente.

Seguindo esse pensamento, a lógica utilizada pela ACO é aproveitada para gerar o presente cálculo de probabilidade ρ para aceitação de uma regra. Onde a probabilidade de aceitação de uma regra (escolher o caminho *i*) está relacionada à taxa de proporção com a qual uma regra semelhante foi aceita. Desse modo, as possibilidades ρ expressas na equação (3.11) se dão através das Fórmulas:

$$\rho_x = TR_x / TP_x \text{ para } TR_x > 0;\ 0{,}8 / TP_x \text{ para } TP_x > 0 \ \&\ TR_x = 0;\ 0{,}8 \text{ em outros casos;}$$
$$x \in \{e,s,se\}. \tag{3.13}$$

A Fórmula em (3.13) é lida como: a probabilidade ρ é igual ao total de regras aceitas *TR* dividido pelo total de regras já propostas *TP*, caso *TR* seja maior que zero; oito décimos dividido por *TP* caso *TP* seja maior que zero e *TR* seja igual a zero; e oito décimos em outros casos.

Percebe-se a semelhança entre as equações (3.13) e (1.14) uma vez que ambas tomam em consideração a quantidade de vezes que um caminho foi escolhido no passado em relação ao total de vezes que foi feita uma escolha de caminho no período equivalente, e seu resultado é um número real no intervalo [0,1].

A ordenação apresentada segue a ideia da heurística ACO com uma pequena adaptação, modelando o problema aqui apresentado para o mais próximo possível do problema original, respeitando suas diferenças. Contudo ambas utilizam mecanismo semelhante para encontrar a viabilidade de escolha de um caminho entre dois disponíveis.

Dando continuidade à busca pelo peso de cada regra a ser ordenada, o total de regras propostas *TP* em que o sensor s aparece e o total de regras aceitas *TR* em que esse sensor aparece (indiferente do valor da leitura) é expresso por:

$$TP_s = \sum_{z=0}^{x} \quad y(z), \text{sendo } y = V_t(i_{s\text{-}},\text{-}) \text{ e } TR_s = \sum_{z=0}^{x} \quad y(z), \text{sendo } y = \Omega(i_{s\text{-}},\text{-}) \qquad (3.14)$$

Onde: o símbolo - significa que não importa o valor assumido por essa variável; *x* é o número total de registros em V_t e Ω para (3.14a) e (3.14b) respectivamente; e *y(a)* assume 1 caso exista um $V_t(i_{s\text{-}},\text{-})$ na posição *a,* e zero caso contrário para (3.14a) e de forma análoga com $\Omega(i_{s\text{-}},\text{-})$ para (3.14b).

Ou seja TP_s é a soma de todos $V_t(i_{s\text{-}},\text{-})$ existentes para todos os *t* existentes, assim como TR_s é a soma de todos $\Omega(i_{s\text{-}},\text{-})$ existentes,

E de forma análoga é feito uma ponderação para o equipamento (indiferente do valor passado para o equipamento). Expresso por:

$$TP_e = \sum_{a=0}^{a=x} \quad y(a) \text{ sendo } y = V_t(\text{-},i_{e\text{-}}) \text{ e } TR_e = \sum_{a=0}^{a=x} \quad y(a) \text{ sendo } y = \Omega(\text{-},i_{e\text{-}}) \qquad (3.15)$$

Ou seja TP_e é a soma de todos os $V_t(\text{-},i_{e\text{-}})$ para todos *t* existentes e TR_e a soma de todos os $\Omega(\text{-},i_{e\text{-}})$existentes. Também de forma análoga, é preciso obter a relação entre um sensor e um equipamento. Esse cálculo é expresso por:

$$TP_{se} = \sum_{a=0}^{a=x} \quad y(a) \text{ sendo } y = V_t(i_{s\text{-}},i_{e\text{-}}) \text{ e } TR_{se} = \sum_{a=0}^{a=x} \quad y(a) \text{ sendo } y = \Omega(i_{s\text{-}},i_{e\text{-}}) \quad (3.16)$$

Por fim é feita uma ponderação da regra, segundo a proximidade geográfica γ entre o sensor e o equipamento, que é dada pela Fórmula:

$\gamma_{se} = 1$ se $(\exists a, L(a,s) \wedge L(a,e))$,

$\gamma_{se} = 0{,}8$ se $(\exists a,b)\ (L(a,s) \wedge L(b,e) \wedge V(a,b))$,

$\gamma_{se} = 0{,}4$ se $(\exists a,b,c)\ (L(a,s) \wedge L(b,e) \wedge (V(a,c) \wedge V(b,c)))$) ,

$\gamma_{se} = 0{,}1 \text{ em outros casos;}$ (3.17)

Onde: *L(x,y)* significa que no cômodo *x* existe o equipamento ou sensor *y* , *V(z,w)* significa que o cômodo *z* é vizinho do cômodo *w*.

Dessa forma, γ_{se} assume um quando *s* está no mesmo cômodo que *e*, oito décimos quando estão em cômodos vizinhos, quatro décimos quando possuem um vizinho em comum e um décimo para qualquer outra possibilidade.

Concluindo, a Fórmula (3.11) representa o peso de cada possível regra a ser proposta. A ponderação feita nessa Fórmula considera que:

a) quanto mais vezes um equipamento *e* estiver presente em uma regra aceita pelo usuário, maior a probabilidade que uma nova regra envolvendo *e* seja aceita pelo usuário;
b) quanto mais vezes um sensor *s* estiver presente em uma regra aceita pelo usuário, maior a probabilidade que uma nova regra envolvendo *s* seja aceita pelo usuário;
c) quanto mais vezes uma regra envolvendo um sensor *s* e um equipamento *e* tiverem sido aceitas pelo usuário, maior a chance que uma nova regra envolvendo *e* e *s* também seja aceita; e
d) quanto mais próximo geograficamente um equipamento estiver de um sensor, maior a probabilidade de que uma regra envolvendo os dois seja aceita pelo usuário.

Justifica-se os pesos aplicados a cada uma das três probabilidades como:

VT_e recebe duas vezes seu valor (ou seja, será responsável por 20% do peso da regra). Seguindo a lógica de que se o usuário achou interessante mudar seu estado de acordo com determinada leitura de um sensor, existe grande probabilidade de que ele queira que outra leitura determine um novo estado para esse equipamento.

VT_s recebe três vezes seu valor (ou seja, será responsável por 30% do peso da regra). Tendo em vista que se um sensor foi utilizado como *trigger* para mudar o estado de um equipamento, a probabilidade de que ele esteja relacionado a outros equipamentos é grande. Se você usa um sensor de presença para ligar/desligar uma lâmpada, existe grande chance de que o utilize para outros equipamentos.

VT_{se} recebe cinco vezes seu valor (ou seja, será responsável por 50% do peso da proposta). Uma vez que se existe uma regra relacionando um determinado equipamento com um determinado sensor existe grande chance de que pelo menos mais uma regra será adicionada com a mesma tupla. Em geral, quando se aplica uma regra para ligar um equipamento quando um sensor perceber a presença, existe grande chance de que se crie uma regra para desligar o equipamento quando o sensor não detecta presença.

Por fim, tem-se o valor total *VT* da regra ponderado pela distância geográfica entre os equipamentos, uma vez que sensores e equipamentos mais próximos tendem a estar mais relacionados entre si do que sensores e equipamentos mais distantes.

O **Proponente de regras** atua com base nos dados gerados pelo ordenador de regras, comparando os pesos de cada regra com os valores mínimos definidos para apresentação da proposta para o usuário. Esse valor mínimo é outra variável usada como parâmetro de controle, e também é alterada pelo módulo de qualidade que será explicado adiante. Caso o peso da regra atinja o valor mínimo, a regra é proposta, caso contrário ela é armazenada em uma tabela que atua como uma fila de espera.

Existe um valor mínimo de propostas que devem ser apresentadas ao usuário. Caso as regras cujo peso seja superior ao valor mínimo não atinjam essa meta, o sistema apresenta as regras da fila de espera até que a meta seja atingida.

Por fim, **Qualidade Total**, módulo que é responsável por acompanhar a taxa de aceitação das regras propostas e a quantidade de propostas geradas. Através desse acompanhamento calcula a necessidade de mudar algum parâmetro do sistema, tais como frequência mínima de recorrência para um equipamento ou sensor ser adicionado pelo Agente Simples, o tamanho mínimo do bloco de propostas para ordenação, o peso mínimo de uma regra para ser proposta ao usuário, e a quantidade de recorrência de um sensor e equipamento para o caso de uma regra idêntica já ter sido negada.

Resumindo esse é o módulo responsável por manter as variáveis de controle com valores que possibilitem o sistema obter os melhores resultados possíveis.

Sua execução tem a mesma frequência da ordenação de regras, utilizando da Fórmula (3.10) para isso. Uma vez que entra em execução ocorre uma série de três testes para verificar se os resultados estão dentro de uma faixa aceitável de qualidade.

O primeiro teste feito pelo módulo objetiva a calcular o valor do peso de corte Ξ_a para uma proposta ser apresentada. Ou seja, o valor mínimo de *VT* de uma proposta para que essa seja repassada ao usuário, expresso por:

$$\Xi_a = \frac{(\Xi_z \cdot \beta_z) + (\Xi_n \cdot \beta_n \cdot 0{,}9)}{(\beta_n + \beta_z)} \text{ caso } \Xi_n < \Xi_p,$$

$$\Xi_a = \frac{(\Xi_z \cdot \beta_z) + (\Xi_n \cdot \beta_p \cdot 0{,}7)}{(\beta_p + \beta_z)} \text{ caso } \Xi_n \geq \Xi_p. \quad (3.18)$$

Onde: Ξ_t é o valor de corte no instante *z*, *z* é o instante da última atualização da nota de corte, Ξ_n é a média dos pesos das regras negadas, Ξ_p é a média dos pesos das regras aceitas, β_né

a quantidade de regras negadas, β_pé a quantidade de regras aceitas; β_z é a quantidade de regras usadas para calcular a média em Ξ_z.

Sumarizando o valor da média $\Xi_a$para (3.18a) será a média ponderada entre Ξ_z e nove décimos de Ξ_n, com a ponderação sendo o número de regras usadas para chegar em cada uma dessas médias; enquanto para (3.18b) será a média ponderada entre Ξ_z e sete décimos de Ξ_p, com a ponderação feita da mesma forma de (3.18a).

Os valores “nove décimos” e “sete décimos”, justificam-se em dois aspectos a) evitar que a média fique paralisada devido ao baixo fluxo de novas regras apresentadas, e b) caso a média de regras aceitas esteja menor que a de regras negadas, significa que o número de regras propostas tende a estar abaixo do ideal, dessa maneira deve-se forçar que mais regras sejam apresentadas ao usuário.

O segundo teste e comparações feitas pelo módulo tem a intenção de verificar a quantidade de regras aceitas *ac* e a quantidade de regras negadas *ng* em um determinado instante *b* para cada período *t*. Logo, toda vez que se acumulam dez regras aceitas, ou dez regras negadas em um período o módulo aplica a seguinte Fórmula:

$$\mu_{at}=\mu_{bt}+\Delta_{np},c \;=c_b+\frac{\Delta_{np}}{|\Delta_{np}|} \text{ se } \Delta_{np}\neq 0 \;. \tag{3.19}$$

Onde μ_{at} é o novo valor de recorrência para o período *t*, μ_{bt} é o valor de recorrência do período *t* no instante *b* logo anterior ao cálculo, Δ_{np} é o cálculo gerado pela diferença de regras aceitas e regras negadas para o período *t*, *c* é o novo valor da cota, e c_b é o valor da cota no instante *b*.

Após o sistema aplicar as Fórmulas (3.19) os valores armazenados para ac_{bt} e ng_{bt} recebem zero. Com isso a contagem para fazer a alteração dessas duas variáveis, usadas como parâmetros de controle do sistema, é reiniciada.

Para calcular o reajuste das variáveis de controle supracitadas, usa-se a Fórmula:

$$\Delta_{np}=(\text{int})\ \frac{(ac_{bt}\text{ - }ng_{bt})}{5} \text{se } (ac_{bt} > ng_{bt}) \wedge (ac_{bt}\text{ - }ng_{bt} > 5),$$
$$\Delta_{np}=(\text{int})\ \frac{(\ ng_{bt}\text{-}ac_{bt})}{7} \text{ se } (ac_{bt} < ng_{bt}) \wedge (\ ng_{bt}\text{ -}ac_{bt}\ > 7),$$
$$\Delta_{np}= 0 \text{ em outros casos.} \tag{3.20}$$

A Fórmula (3.20) é lida como: caso ac_{bt} seja maior que ng_{bt}, e ac_{bt} menos ng_{bt} seja maior que cinco, Δ_{np} recebe a parte inteira da divisão por cinco de ac_{bt} menos ng_{bt}; caso ac_{bt}

seja menor que ng_{bt} , e ng_{bt} menos ac_{bt} seja maior que sete, Δ_{np} recebe a parte inteira da divisão por sete de ng_{bt} - ac_{bt} ; e Δ_{np}recebe zero para qualquer outra situação.

Por fim o terceiro teste é feito a cada fechamento de período, ou seja, se o período em consideração é “dia” sempre que um dia termina esse teste é executado. Quando esse teste entra em execução ele compara a quantidade de propostas Q_{bt} do período t no instante b, com a quantidade mínima L_t de regras esperadas, e a quantidade máxima H_t de regras esperadas. Caso i) Q_{bt} seja menor que L_t , ou ii) Q_{bt} seja maior que H_t aplica-se a Fórmula (3.19a); sendo o cálculo de Δ_{np}feito pela Fórmula (3.21) para o primeiro caso e pela Fórmula (3.22) para o segundo. Se Q_{bt} estiver dentro do intervalo fechado L_t a H_t nem um ajuste é feito.

A Fórmula para controle do primeiro caso supracitado é expressa por:

$$\Delta_{np} = \text{-(int)}\ \frac{H_t}{5 \cdot L_t}\ \text{se}\ Q_t = 0,$$
$$\Delta_{np} = \text{-(int)}\ \frac{L_t}{5 \cdot Q_t}\ \text{se}\ Q_t > 0\ . \qquad (3.21)$$

Sendo lida como Δ_{np} recebe o valor inteiro negativo da divisão de H_t por cinco vezes o valor de L_t, caso Q_t seja zero; ou recebe o valor inteiro negativo da divisão de L_t por cinco vezes o valor de Q_t caso o último seja maior que zero.

Enquanto a Fórmula para controle do segundo caso é expressa por:

$$\Delta_{np} = (\text{int})\ \frac{Q_t}{H_t}\ . \qquad (3.22)$$

Sendo lida como Δ_{np} recebe o valor inteiro da divisão de Q_t por H_t.

3.3.4. Ambiente virtual para simulação

Para a validação deste trabalho não é necessário a existência de uma representação gráfica dos objetos desse mundo virtual, uma vez que o foco é uma análise cientifica, a observação e comparação dos dados de entrada e saída seriam suficientes. Entretanto, essa representação, como afirmado por Dias e Carvalho (2007), facilita a observação do ambiente e suas variáveis.

O ambiente virtual neste trabalho segue o conceito de visualização científica, ficando restrito às necessidades de representação exata das respostas do ambiente às interações. O

ambiente simula uma casa e os objetos necessários para avaliação das interações que serão feitas. Portanto o ambiente virtual será apenas uma representação em 2D.

A modelagem da casa e dos seus objetos, apresentam-se por um código HTML configurado para exibir imagens representativas desses objetos nos locais onde se fazem presente. Um navegador é usado para exibição desse ambiente de simulação.

O ambiente virtual representa as ações das pessoas dentro dele, sendo estas pré-estabelecidas como rotinas de teste. Em síntese, cada pessoa criada nesta simulação terá suas identificações pré-estabelecidas assim como ações que serão definidas para simular as possíveis interações em uma casa real. A simulação finda avaliar a atuação do *Framework* AIHC e do agente inteligente em relação a rotinas estabelecidas, validando o conjunto dos módulos sem a necessidade de instalar os periféricos para os testes em um ambiente real, reduzindo significativamente o custo para validação do software.

A concepção de agentes inteligentes para interação com o ambiente é de suma importância para validação do funcionamento do sistema AIHC, *Framework* e Inteligência artificial. Esses foram implementados no modelo Agente Reativo, possuindo uma tabela de ações pré-determinadas, as quais tencionam simular pessoas na casa virtual.

Por fim é preciso uma Conexão com o CDB, de modo que permita fazer a leitura dos estados de cada objeto e representá-los no navegador.

3.4. MÉTODO DE CONSTRUÇÃO

O método de construção seguiu a lógica abordada nas seções anteriores. Primeiro foram gerados os dois bancos de dados (AIHC.sql e knowledge.sql) e em seguida o CRUD (banco.h). Este enfoque possibilita realizar todos os testes de cada funcionalidade separada, evitando a construção de todo o sistema, para apenas depois fazer as correções necessárias. Este tipo de abordagem contribui para que não haja retrabalho relacionado a modificar uma função.

Em seguida o *Framework* foi construído de forma incremental, para possibilitar os testes de cada nova funcionalidade criada. Ou seja, cada novo módulo era implementando junto de suas funções para comunicar-se com o *Framework* (integração).

Construiu-se o cadastro para usuários, crucial para definir-se o *Master User* (M.U.) o qual todos os outros cadastros são dependentes. Ressaltando que esse primeiro usuário foi adicionado diretamente no banco de dados, uma vez que não é possível seguir as regras iniciais de construção, devido a inexistência de um sensor registrado para validar seus dados biométricos e um M.U. para validação.

O cadastro para Sensor foi criado em seguida, possibilitando o reconhecimento de novos usuários. Após a implementação deste módulo e o a adição do primeiro sensor

biométrico, testes do sistema sem nenhuma intervenção direta no banco de dados tornam-se possíveis. Na sequência implementou-se os cadastros de Equipamentos e de Atuadores, e por fim o *Kernel*, o Controle e o “Interface com Usuário”, completando com isso o *Framework* AIHC e possibilitando testes globais do mesmo.

Cabe ressaltar que, com o esqueleto do sistema construído e as funções principais testadas, construiu-se o ambiente virtual com intuito de simular uma casa para validação do sistema em um ambiente o mais próximo possível da realidade.

Com o *Framework* implementado e testado, foi implementado o Agente Inteligente. Esse foi construído seguindo a mesma ordem apresentada na Seção 3.3.3. Cada um desses módulos prepara a informação para o módulo seguinte, facilitando o entendimento da ordem de precedência de construção visando testes individuais de cada módulo.

4. IMPLEMENTAÇÃO

Seguindo a linha de raciocínio do Capítulo anterior, este Capítulo apresenta as principais funções usadas na construção do sistema. Começando pelos bancos de dados, seguindo pelas funções do *Framework* e por fim apresentando a Inteligência Artificial.

4.1. BANCO DE DADOS

A implementação dos bancos seguiu o modelo de estrutura de tabelas apresentadas nas Figuras 2.4 e 2.5. Com as tabelas e as relações necessárias criadas, construiu-se um CRUD, possibilitando todas as interações do *Framework* com o banco de dados. Implementou-se esse CRUD em um formato genérico, provendo as consultas básicas ao sistema.

O CRUD foi construído em uma biblioteca denominada banco.h, responsável pelas principais *query's*(pesquisas) SQL relativas a cadastrar, atualizar, consultar e apagar os registros. Objetivando exemplificar apresentam-se as seguintes *query's*:

a) "consutlaSimples", é uma *querry* utilizada para pesquisas onde apenas um registro será retornado. Um exemplo para isso é: "qual usuário é dono de um determinado *template* de digital";
b) "consultaVetor", é semelhante a "consultaSimples", com a exceção de que o retorno nesse caso serão múltiplos registros. Para exemplificar: "quais equipamentos estão ligados em um determinado cômodo";
c) "upDateSimples", é uma função que irá atualizar apenas um campo de um determinado registro, tal como o estado do equipamento de id 2 por exemplo;
d) "deleteSimples", é responsável por apagar um registro específico. Como um equipamento que será retirado da casa, logo um registro de id específico será apagado.

Todas essas funções são semelhantes entre si, excetuando os múltiplos do simples que possuem um retorno diferente. No mais a única diferença é a consulta que é enviada para o BD. Assim, a Figura 3.1 que apresenta a "consultaTotal" serve para representar o código de todas as *querry's* simples enquanto a Figura 3.2 que apresenta "consultaVetor" serve de exemplificação para todas as *querry's* de múltiplos registros.

```
consultaTotal(BD,campoResposta,tabela,coluna,condicao){
 consulta;
 MYSQL_RES *result; //variável que recebe o resultado
 MYSQL_ROW dados; //variável que recebe os dados
 str1="select "+campoResposta+" from "+tabela+" where "+coluna+" = '"+condicao+"';";
     if(mysql_query(BD,str1.c_str())){return mysql_error(BD);} //caso exista erro, retorna qual erro encontrado
 mysql_query(BD,str1.c_str());
 result = mysql_store_result(BD);// Recebe os dados da cunsulta
  if (result){ // Se consultou (sem erros)
    if ((dados=mysql_fetch_row(result)) != NULL){  // Enquanto receber dados vai escrevendo
    istringstream buffer(dados[0]); // Cria a variável que recebera a string a ser convertida
    buffer >> consulta; }  // Carrega a resposta que o usuario procura na variavel de resposta(consulta)
    mysql_free_result(result);// Limpa da memória
  return consulta;} //retorna a consulta feita
  else
  {return "";}//caso a consulta seja vazia retorna uma string vazia
}
```

Figura 3.1: Consulta Simples Fonte: Elaborado pelo autor

```
consultaVetor(BD,campoResposta,tabela,coluna,condicao){
  MYSQL_RES *result;
  MYSQL_ROW dados;
  str1="select "+campoResposta+" from "+tabela+" where "+coluna+" = '"+condicao"';";
  if(mysql_query(BD,str1.c_str())){return mysql_error(BD);}
    mysql_query(BD,str1.c_str());result = mysql_store_result(BD);
  if (result){
    if ((dados=mysql_fetch_row(result)) != NULL){ //caso exsita pelo menos um registro compativel com a consulta
     istringstream buffer(dados[0]); buffer >> auxiliar;
     consulta = consulta+auxiliar; //adiciona o primeiro valor no vetor
    while ((dados=mysql_fetch_row(result)) != NULL) { //caso exista mais de um registro na consulta efetuada
     istringstream buffer(dados[0]);
     buffer >> auxiliar; // carrega a variavel auxiliar com o registro da consulta
     consulta = consulta+","+auxiliar; //concatena o proximo registro na variavel de resposta separados por virgula
    }
   }
     mysql_free_result(result);
     return consulta;
   }
     else{return "";}//consulta retornou vazia
}
```

Figura 3.2: Consulta Múltipla Fonte: Elaborado pelo autor

Construiu-se o CRUD visando atender as demandas de ambos os bancos, tanto o CDB (AIHC.sql) quanto o KDB(knowledge.sql). Salienta-se que a biblioteca banco.h é utilizada como forma de acesso para os dois bancos possuindo uma função específica de comunicação para cada, responsáveis por criar a conexão entre o sistema e o BD.

4.2. *FRAMEWORK*

A fundamentação do *Framework* deu-se pela implementação de uma função de acesso a cada estrutura a ser utilizada. Ou seja, através do CRUD genérico criou-se um CRUD específico para pessoa, cômodo, sensor, equipamento, atuador, regras específicas (regras de automação), leituras dos sensores (listaacao) e mudança de estado dos equipamentos (comandos). Todos esses CRUD's específicos são semelhantes entre si, diferindo-se apenas pelo nome das tabelas e campos. Dessa forma a Figura 3.3 que apresenta as funções do CRUD pessoas, é usado para exemplificar todos os outros CRUD's específicos.

```
//Read from id
consultaNomePessoa(BD,id){return consultaTotalString(BD,"nome","pessoa","id",id);}
consultaTelefonePessoa(BD,id){return consultaTotalString(BD,"telefone","pessoa","id",id);}
consultaEmailPessoa(BD, string id){return consultaTotalString(BD,"email","pessoa","id",id);}
consultaDigitalPessoa(BD,string id){return consultaTotalString(BD,"digital","pessoa","id",id);}
consultaFacePessoa(BD,id){return consultaTotalString(BD,"face","pessoa","id",id);}
consultaVozPessoa(BD,id){return consultaTotalString(BD,"voz","pessoa","id",id);}
consultaNivelAcessoPessoa(BD,id){return consultaTotalString(BD,"nivelAcesso","pessoa","id",id);}
//Read from biometria
consultaDigital(BD,digital){return consultaTotalString(BD,"id","pessoa","digital",digital);}
consultaFace(BD,face){return consultaTotalString(BD,"id","pessoa","face",face);}
consultaVoz(BD,voz){return consultaTotalString(BD,"id","pessoa","voz",voz);}
//UPDATE from id
alteraTelefonePessoa(BD,id,novo){return upDateTotalString(BD,"telefone","pessoa","id",id,novo);}}
alteraEmailPessoa(BD,id,novo){return upDateTotalString(BD,"email","pessoa","id",id,novo);}}
alteraNivelAcessoPessoa(BD,id,novo){return upDateTotalString(BD,"nivelAcesso","pessoa","id",id,novo);}}
alteraNomePessoa(BD,id,novo){return upDateTotalString(BD,"nome","pessoa","id",id,novo);}}
//Create
  criaPessoa(BD,nome,telefone,email,digital,face,voz,acesso){
   camposIns ="nome,telefone,email,digital,face,voz,acesso";
   valoresIns = nome+","+telefone+","+email+","+digital+","+face+","+voz+","+acesso;
   return inserTotalString(BD,"pessoa",camposIns,valoresIns);
   }
// Delete from id
apagaPessoa(BD,id){return deleteTotalString(BD,"pessoa","id",id);}
```

Figura 3.3: Conexão de Pessoas ao CRUD Fonte: Elaborado pelo autor

Os atributos utilizados para construção dos cadastros supracitados e do CRUD referente a lista ação e a lista de mudança de estados dos equipamentos (comandos) são os mesmos representados nas suas respectivas tabelas presentes na Figura 2.4.

Seguiu-se então para a implementação do módulo cadastro e rotinas foram criadas para preencher os campos de cada tipo de registro conforme subseção 3.3.1.

Ressalta-se uma peculiaridade do módulo de cadastro, o cadastro de Atuador, que por herdar muitas características de equipamentos utiliza duas tabelas para ser cadastrado. As duas tabelas citadas são a tabela de atuador e a de equipamento, complementarmente existe uma chave estrangeira ligando a primeira com a segunda, uma vez que o atuador é um tipo de equipamento. Isso surge da necessidade de uma representação diferenciada dessa estrutura na visualização. Pela mesma razão a estrutura “porta”, sendo um tipo específico de atuador, possui relação em três tabelas: “porta”, “atuador” e “equipamento”.

A função construída em seguida é o acionamento de sensor, a qual envia os dados coletados do sensor para a lista de ação, e que posteriormente é lida pelo *Kernel* do *Framework*. Essa função está representada na coluna Drive da Figura 2.1 com o nome Sensor e a Figura 3.4 apresenta o código utilizado para sua implementação.

```
acionaSensor(){
 while(true){
  buffer = leituraSensor(_idEquipamento);
  acessaBanco(BD);
   if(buffer = "exit"){break;} //confere se o comando de saida do sistema foi acionado
   else
   if(buffer = "desliga"){desligaSensor(BD,_idEquipamento);} //desliga sensor
   else
   if(buffer = "liga"){ligaSensor(BD,_idEquipamento);}  //liga sensor
   else{ // caso seja outro comando (que não liga/desliga/exit)
     alteraLeituraSensor(BD,buffer,_idEquipamento); //atualiza a entra de inteiros do sensor
     insereLeituraListaAcao(BD,buffer,_idEquipamento);
   }
  fechaCon(&DBCon);
 }
}
```

Figura 3.4: Acionamento Sensor Fonte: Elaborado pelo autor

Em seguida implementou-se uma função específica para envio de comando e atualização de estado, análoga ao visto na Figura 3.4 e está representada na Figura 2.1 na coluna “drive”, como uma das partes do “Ouvido Externo”. Sua utilidade é enviar um comando de mudança de estado para o CDB para que o Controle do *Framework* processe esse comando realizando a mudança de estado.

Após a construção de toda essa estrutura, o módulo *Kernel* foi implementado. Esse é o mais complexo e de maior importância dentro do *Framework*, uma vez que o “regras específicas” trata as regras de automação. Assim, ocorre um processamento de entradas dos sensores, onde, dado uma lista de entradas, é feita uma análise para uma tomada de decisão. A função responsável por isso é vista na Figura 3.5. Um *loop* infinito é responsável por manter a leitura de todos os novos registros da tabela “listaacao”, onde estão armazenadas todas entradas dos sensores.

```
controlaAcao(){
  id=1; //inicia o id para buscar na lista de ação
  while(true){ //executa a busca na lista de ações enquanto o programa estiver ativo
  acessaBanco(BD);
  aguardaEntrada(bd,id);
  fechaSinal(BD); // fecha o semafaro para que outros programas não acessem o banco
  idSensor = consultaIdSensorAcao(&BD,idc); // consulta qual o id do sensor passando informação
     if(sensorBiometrico(idSensor)){
     comodo = getComodoSensor(BD,idSensor);
     atualizaPessoasComodo(BD,comodo,leitura);
     }
     if(existeRegra(BD,idSensor,leituraSensor)){//pesquisa se existe regra para o sensor que enviou a leitura
     executaRegra(BD,leituraSensor,idSensor); //executa a regra caso a leitura e id correspondam a uma regra
     }
    id++;  // passa para o proximo id
    abreSinal(BD); // apre o sinal liberando o banco para outros modulos
    Sleep(200);   // pausa para facilitar a apresentação do funcionamento do programa
  fechaCon(BD);
  }
}
```

Figura 3.5: *Kernel* - Processamento de Entradas Fonte: Elaborado pelo autor

A leitura de ações acontece em um processo contínuo, lendo as ações registradas no banco e ficando em um estado de espera (aguardaEntrada()) quando todos os registros salvos foram lidos. Logo que uma nova entrada é registrada, o sistema sai do estado de espera e processa a(s) nova(s) entrada(s) do banco.

Toda entrada é testada pelo existeRegra(). Caso exista uma regra específica equivalente a leitura feita pelo sensor, o executaRegra() cria um registro de comando no CDB para mudança de estado do equipamento de acordo com a tabela de regras específicas. Esse comando segue a tabela de regras específicas para definir o novo estado do equipamento.

A seleção de regras para execução segue o modelo do código apresentado na Figura 3.6. Onde o sistema pega a lista de regras existentes para a tupla (leitura,sensor), faz um teste pelo comparaRegra() e encaminha cada uma das regras aprovadas para o executaRegrasIndividual(), que adiciona um registro na tabela comando para mudança de estado do devido equipamento.

```
executaRegra(BD,leituraSensor,idSensor){//recebe a string com leitura do sensor e o id do sensor
 bufferVetor = listaIdRegras(BD,idSensor,leituraSensor); // recebe a lista das regras para id do sensor
 while(existeIdLista(bufferVetor)){ // verifica se tem algum id na regra
   buffer = idDaLista(bufferVetor); // recebe o primeiro id da lista de id
   testa = comparaRegra(BD,leituraSensor,idSensor,buffer);//verifica se a leitura do sensor aciona a regra do id no buffer
   if(testa){ // caso acione a regra teste tera valor 1, caso contrario valor sera 0
     executaRegraIndividual(BD,buffer,leituraSensor,idSensor); // envia o id (buffer) a leitura e id do senor para executar a regra
   }   // fim do if (regra deve ser acionada)
   bufferVetor = novaLista(bufferVetor);// remove o primeiro id da lista de id's
 } // fim do while existe id no buffer vetor
}
```

Figura 3.6: Tomada de decisão Fonte: Elaborado pelo autor

O “compara regra” que é visto na Figura 3.7, atua recuperando o tipo de comparação a ser feita (tipo) na regra (idc) e o valor a ser comparado (valor), em seguida passa esses valores e a leitura do sensor para a função comparaValorRegra(). Seu retorno é o mesmo retornando pelo comparaValorRegra.

```
comparaRegra(BD, leituraSensor, idSensor, idc){
  tipo = consultaTipoComparacaoRegra(BD,idc); // recebe o tipo de comparação a ser usado
  valor = consultaValorComparadoRegra(BD,idc); // recebe o valor a ser comparado
  result = comparaValorRegra(tipo,valor,leituraSensor); // compara o valor da leitura com o valor da regra, segundo o tipo definido
  return result; // retorna 1 caso o valor atenda a especificação de comparação e 0 caso contrario
}
```

Figuras 3.7: Comparação de Regra Fonte: Elaborado pelo autor

O “compara valor de regra”, presente na Figura 3.8, é utilizado para a comparação dos valores definidos na regra com os valores lidos pelo sensor. Sua finalidade é definir se o valor lido pelo sensor está dentro do esperado para ativar a regra específica.

```
comparaValorRegra( tipo, valor, valor2){
  if(tipo == 1){
   if(valor < valor2){return true;}
   else return false;
  }
  if(tipo == 0){
   if(valor == valor2){return true;}
   else return false;
  }
  if(tipo == -1){
   if(valor > valor2){return true;}
   else return false;
  }
}
```

Figuras 3.8: Compara Valor Regra Fonte: Elaborado pelo autor

Na Figura 3.9 é visto o código para o executaRegraIndividual(), após a função receber o id equivalente ao registro de uma regra são carregas as variáveis referente ao id do equipamento (idEquipamento) e ao valor para seu novo estado(buffer). Estes são os

parâmetros passados na chamada da função enviaComandoLista(). Essa última função atua criando um registro na tabela comando com o novo estado do equipamento especificado na regra (buffer).

```
executaRegraIndividual(BD,id){
  tipoeq = consultaTipoEquipamento(DBCon,id); // consulta o tipo de equipamento a ser auterado
  idEquipamento = consultaIdEquipamento(DBCon,id); // recebe o id do equipamento a ser alterado
  buffer = consultaValorPassado(DBCon,id); // consulta o valor a ser passado para o equipamento
  enviaComandoLista(idEquipamento,buffer);//coloca na lista comando o comando enviado
}
```

Figura 3.9: Execução de Regra Fonte: Elaborado pelo autor

O módulo de Controle, faz a leitura da tabela “comando”, preenchida pelo *Kernel* e por entradas diretas do usuário, essa última através do ouvido externo. Alterando o estado do equipamento segundo os dados contidos no registro, assim como altera o estado do devido equipamento no CDB na tabela referente.

Por fim a Figura 3.10 representa a formação de uma Regra Específica com informações fornecidas pelo M.U. Esse processamento acontece quando um “ouvido externo” é acionado no modo cria regra. A frase enviada pelo usuário é processada nessa função, que gera uma nova regra.

```
listenRegra(frase,BD){
  tabela="regrasespecificas"; //tabela que será modificada
  campoIns= getNomeCamposTabela(tabela);
  valor=separaRegra(frase); // faz a conversão da frase enviada pelo usuario para comando sql
  insereDados(BD,tabela,campoIns,valor); // cria a regra no banco
}
```

Figura 3.10: Criação de regra Fonte: Elaborado pelo autor

Essa função é análoga para o cadastro de regras específicas vindas de outros módulos, tal como a Inteligência Artificial.

Impende ressaltar que a “interface usuário” faz uso de uma tabela exclusiva para isso no banco de dados e de um semáforo de comando e resposta, simulando um *socket* de comunicação.

4.3. INTELIGÊNCIA ARTIFICIAL

O módulo Agente Inteligente é responsável por analisar as ações e comandos recorrentes e definir quais novas regras devem ser propostas ao M.U. como possíveis regras a serem adotadas.

A A.I. foi implementada para atender a classificação de agente inteligente orientado a utilidade, contínuo, comunicativo com aprendizagem. Para isso utilizou-se grande fundamentação na metaheurística ACO apresentada na Seção 2.5 e cuja relação é descrita nas fórmulas apresentadas na subseção 3.3.3.

A ordem utilizada para implementar o módulo de inteligência artificial foi a mesma utilizada na apresentação da modelagem, com a adição de uma função para inicializar os módulos. O detalhamento do módulo agente inteligente pode ser consultado nos Anexos.

4.4. Ambiente de Simulação

Para testar o funcionamento do sistema de forma integral foi implementado um ambiente virtual usando HTML e JavaScript, representando uma casa e todos os objetos ligados ao *Framework* AIHC. A Figura 4.1 apresenta uma instância da casa feita para simulação.

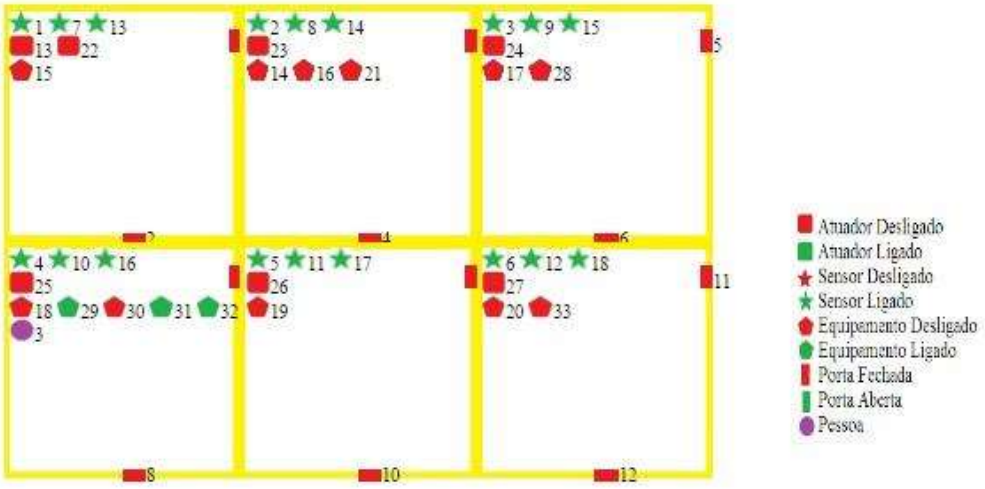


Figura 4.1: Casa Simulada Fonte: Elaborado pelo autor

Essa imagem é a representação da casa utilizada para a simulação. Consiste em uma casa de seis cômodos, cada um com uma variedade de sensores e equipamentos. Cada estrela corresponde a um sensor, cada quadrado a um atuador, cada pentágono representa um equipamento, enquanto os retângulos representam as portas e os círculos as pessoas. Além disso, suas cores definem seus estados, sendo verde para ligado e vermelho para desligado. O código busca os dados no CDB e os representa em uma página web.

Além da representação visual, foram implementados alguns agentes de teste para replicar a ação humana dentro de uma casa, seguindo o modelo de agente inteligente reflexivo, conforme explicado por Russell e Norvig (1995). Ou seja, eles possuem uma tabela na qual dada uma instância do mundo existe uma resposta já selecionada.

Esse agente de teste baseia sua interação com o ambiente na hora real. A descrição de suas interações será apresentada no próximo Capítulo. Suas duas principais funções são movimento() e atuacao(). A primeira define a movimentação de um agente entre os cômodos enquanto o segundo define um comando enviado pelo agente para o *Framework*.

```
mudaComodo(idS1,idS2,idPessoa){
  DBCon;
  acessaBD(BD);
  _face = consultaFacePessoa(BD,idPessoa);
  mudaComodoF(BD,idS1,idS2,_face);
  fechaCon(BD);
  Sleep(20);
}
```

Figura 4.2: Função Movimento Fonte: Elaborado pelo autor

A função na Figura 4.2 é utilizada para simular a movimentação de uma pessoa dentro da casa. Um exemplo de movimentação é “a pessoa 3 entra no cômodo 4 vindo de fora da casa”. Após essa movimentação um círculo roxo com o número 3 irá aparecer no cômodo 4, da forma como é visto na Figura 4.1.

```
usaEquipamentoF( id, valor, idPessoa){
  tempo = idPessoa * 500;
  Sleep(tempo);
  buffer = endereco("equipamento",id,valor,idPessoa);
  system(buffer);
}
```

Figura 4.3: Função Atuação Fonte: Elaborado pelo autor

O código da Figura 4.3 é uma função para simular a interação humana com os equipamentos da casa. Um exemplo de interação é fechar uma porta por exemplo, o que na representação em HTML significa mudar a cor dessa porta de verde para vermelho.

Após a implementação de todas essas funções, classes e tabelas dos bancos o sistema está pronto para funcionar virtualmente, controlando todos os equipamentos registrados, e tratando todas as interações feitas por pessoas/ambiente. Também estará apto a melhorar sua suposição de novas regras, aprendendo à medida que estas forem aceitas ou negadas pelo(s) M.U.

5. TESTES E RESULTADOS

Este Capítulo apresenta a simulação feita para testar e validar o sistema desenvolvido no Capítulo anterior e avaliar os resultados obtidos. O *Framework* AIHC produzido nesse estudo foi testado com sucesso, através de envio de comandos e leituras de sensores, que foram observadas pelo ambiente virtual. O *Framework* apresentou um funcionamento correto para todos os comandos enviados, assim como as respostas corretas para todas as leituras de sensores.

Após esses testes, duas simulações foram realizadas com foco na avaliação do desempenho e aprendizagem da Inteligência Artificial utilizada. Nas próximas seções serão apresentadas a descrição das simulações realizadas assim como os resultados obtidos.

5.1. Simulação

A simulação teve como objetivo replicar a interação de pessoas com objetos da casa, a fim de avaliar o desempenho da inteligência artificial em deduzir possíveis regras de automação. Foram realizadas duas simulações utilizando de uma mesma residência virtual. Cada simulação teve um agente de teste diferente, cada qual com uma rotina pré-definida de movimentos e comandos.

O modelo de residência utilizado para as simulações é o mesmo apresentado na Figura 4.1. Ou seja, uma residência de seis cômodos possuindo 18 sensores e 33 equipamentos, esse último grupo é subdividido em 12 portas, sete atuadores e 14 equipamentos genéricos. A lista contendo cada sensor é vista na Tabela 1.1 enquanto a lista com os equipamentos é vista na Tabela 1.2.

Lista de Sensores								
id	tipo	nome	id	tipo	nome	id	tipo	nome
1	incendio	incendio1	7	biometrico	face 1	13	generico	presença 1
2	incendio	incendio 2	8	biometrico	face 2	14	generico	presença 2
3	incendio	incendio 3	9	biometrico	face 3	15	generico	presença 3
4	incendio	incendio 4	10	biometrico	face 4	16	generico	presença 4
5	incendio	incendio 5	11	biometrico	face 5	17	generico	presença 5
6	incendio	incendio 6	12	biometrico	face 6	18	generico	presença 6

Tabela 1.1: Lista de Sensores Fonte: Elaborado pelo autor

Lista de Equipamentos								
id	tipo	nome	id	tipo	nome	id	tipo	nome
1	porta	porta 1	12	porta	porta 12	23	atuador	extintor 2
2	porta	porta 2	13	atuador	ar condicionado	24	atuador	extintor 3
3	porta	porta 3	14	genérico	TV	25	atuador	extintor 4
4	porta	porta 4	15	genérico	lâmpada 1	26	atuador	extintor 5
5	porta	porta 5	16	genérico	lâmpada 2	27	atuador	extintor 6
6	porta	porta 6	17	genérico	lâmpada 3	28	genérico	rádio
7	porta	porta 7	18	genérico	lâmpada 4	29	genérico	rádio 2
8	porta	porta 8	19	genérico	lâmpada 5	30	genérico	geladeira
9	porta	porta 9	20	genérico	lâmpada 6	31	genérico	fogão
10	porta	porta 10	21	genérico	tv	32	genérico	cafeteira
11	porta	porta 11	22	atuador	extintor 1	33	atuador	ar condicionado

Tabela 1.2: Lista de Equipamentos Fonte: Elaborado pelo Autor

Os sensores dividem-se em três grupos a) ids 1 a 6, b) ids 7 a 12 e c) ids 13 a 18. O primeiro grupo representa sensores para detecção de incêndio. O segundo grupo são sensores biométricos de reconhecimento facial. Enquanto o terceiro são sensores genéricos de indicação de presença.

Os equipamentos também distinguem-se em três grupos a) ids 1 a 12, b) ids 13, 22 a 27 e 33 e c) 14 a 21 e 28 a 32. O primeiro grupo são portas, ou seja, são ao mesmo tempo equipamentos, atuadores e porta. O segundo grupo são atuadores, ou seja, são ao mesmo tempo equipamentos e atuadores. Sendo os de id 13 e 33 do tipo ar condicionado enquanto os de id 22 a 27 extintores de incêndio de teto. O terceiro grupo representa os equipamentos genéricos, dos quais os ids 15 a 20 são lâmpadas, os ids 14 e 21 televisores, os ids 28 e 29 rádios, o id 30 geladeira, o id 31 fogão e o id 32 cafeteira.

Ao associar as Tabelas 1.1 e 1.2 com a Figura 4.1, percebe-se a disposição desses equipamentos e sensores pelos seis cômodos da casa virtual concebida para essa simulação.

Com a implementação da casa virtual, desenvolveu-se um agente para cada simulação e cada um com uma rotina diferente. Essas rotinas repetem-se diariamente e estão expressas nas Tabelas 2.1 e 2.2.

Agente de teste 1			
horário	movimento/comandos	horário	movimento/comandos
06:00	C(8,1), M(0,4), C(8,0), C(18,1), C(32,1), C(29,8)	13:00	C(18,0), C(29,0), C(8,1), M(4,0), C(8,0)
07:00	C(32,0), C(29,5)	18:00	C(5,1), M(0,3), C(5,0), C(28,8), C(17,1)
08:00	C(9,1), C(18,0), C(29,0), M(4,5), C(19,1), C(9,0)	19:00	C(28,5)
09:00	C(4,1), C(19,0), M(5,2), C(4,0), C(14,7)	20:00	C(3,1), C(28,0), C(17,0), M(3,2), C(3,0), C(16,1), C(14,7)
10:00	C(14,10)	21:00	C(14,10), C(16,0)
11:00	C(14,0), C(4,1), M(2,5), C(4,0), C(5,1), M(5,4), C(18,1), C(31,1), C(29,8)	22:00	C(14,0), C(1,1), M(2,1), C(1,0), C(13,17)
12:00	C(31,0), C(29,5)	05:00	C(2,1), C(13,0), M(1,4), C(2,0), C(8,1), M(4,0), C(8,0)

Tabela 2.1: Agente de teste 1 Fonte: Elaborada pelo autor

Agente de teste 2			
horário	movimento/comandos	horário	movimento/comandos
18:00	C(10,1), M(0,5), C(10,0), C(4,1), M(5,2), C(4,0), C(16,1), C(14,2)	22:00	C(19,0), C(7,1), M(5,4), C(28,0), C(7,0), C(2,1), M(4,1), C(2,0), C(13,20)
19:00	C(14,7)	06:00	C(13,0), C(2,1), M(1,4), C(2,0), C(31,1), C(32,1)
20:00	C(14,0), C(16,0), C(4,1), M(2,5), C(4,0), C(7,1), M(5,4), C(7,0), C(18,1), C(31,1), C(28,1)	07:00	C(31,0), C(32,0), C(8,1), M(4,0), C(8,0)
21:00	C(18,0), C(31,0), C(7,1), M(4,5), C(7,0), C(19,1)		

Tabela 2.2: Agente de teste 2 Fonte: Elaborada pelo autor

Assim as Tabelas 2.1 e 2.2 entendem-se como, dado um horário o agente faz uma sequência de movimentações e comandos. Assim as tuplas *M(x,y)* representam uma movimentação do cômodo *x* para o cômodo *y*, sendo zero a representação de fora da residência. Enquanto a tupla *C(i,j)* representa que o equipamento *i* teve seu estado alterado para o valor *j*, sendo o valor zero a representação de que o equipamento foi desligado.

Em seguida, definiu-se um conjunto de regras esperadas para cada teste. Esse conjunto representa quais as regras são esperadas que o sistema encontre e proponha ao usuário. As Tabelas 3.1 e 3.2 representam as regras para a primeira e segunda simulação

respectivamente. A regra pode ser interpretada como, o Tipo de Comparação refere-se se o valor de leitura do sensor, de Id equivalente ao salvo em Id Sensor, deve ser igual (0), maior (1) ou menor (-1) que o valor a ser comparado (Valor Comparado) salvo na regra. Caso essa comparação seja verdadeira, o equipamento de id equivalente ao Id Equipamento recebe o valor salvo em Valor Passado.

Regras para o Simulação 1					
N	**Tipo Comparação**	**Id Sensor**	**Valor Comparado**	**Id Equipamento**	**Valor Passado**
1	0	10	12	18	1
2	0	10	12	29	8
3	0	10	0	8	0
4	0	10	0	18	0
5	0	10	0	29	0
6	0	8	12	14	7
7	0	8	0	14	0
8	0	9	12	28	8
9	0	9	12	17	1
10	0	9	0	28	0
11	0	9	0	17	0
12	0	7	12	13	17
13	0	7	0	13	0

Tabela 3.1: Regras da Simulação 1 Fonte: Elaborado pelo autor

Regras para o Simulação 2					
N	**Tipo Comparação**	**Id Sensor**	**Valor Comparado**	**Id Equipamento**	**Valor Passado**
1	0	16	0	31	0
2	0	14	0	14	0
3	0	17	0	19	0
4	0	13	2	13	20
5	0	16	0	18	0
6	0	13	0	13	0
7	0	16	0	32	0
8	0	16	0	8	0

Tabela 3.2: Regras da Simulação 2 Fonte: Elaborado pelo autor

Para definir-se as regras contidas nas Tabelas 3.1 e 3.2, analisou-se as rotinas das Tabelas 2.1 e 2.2 escolhendo regras que se demonstravam razoáveis de um usuário real desejar adotar.

A condição de sucesso considerada para essas simulações é que o sistema detecte e proponha todas as regras presentes nas Tabelas 3.1 e 3.2. Sendo a Tabela 3.1 a condição de sucesso para a primeira simulação e a Tabela 3.2 a condição de sucesso para a segunda simulação.

Para proporcionar maior agilidade às simulações, foram usados dois aplicativos simples:

- O primeiro teve como objetivo adiantar a hora do relógio a cada cinco minutos, permitindo que cada simulação tivesse sua duração reduzida de semanas para poucos dias.
- O segundo aplicativo foi responsável por responder todas as propostas apresentadas ao M.U., aceitando as regras presentes nas Tabelas 3.1 e 3.2, dependendo da qual simulação, e rejeitando as demais. Possibilitando, com isso, simulações sem um monitoramento constante para responder as regras propostas pelo sistema.

Finalizando os cenários de simulação, definiu-se os parâmetros iniciais das variáveis de controle. Esses valores são vistos na Tabela 4 e foram aplicados às duas simulações.

Parâmetros Iniciais												
Cota	mínimo para apresentação diária	frequência			qualidade						Corte	
					Dia		Mês		Ano		media	qntd
		Dia	Mês	Ano	Min	Max	Min	Max	Min	Max		
3	7	2	5	10	1	10	30	300	360	3600	0	0

Tabela 4: Parâmetros Iniciais Fonte: Elaborado pelo autor

Esses parâmetros representam as variáveis de controle apresentada no Capitulo 3, sendo:

- O valor em cota, representa a variável c,
- O mínimo para apresentação diária, representa a meta de propostas diárias, a qual decide se uma regra com peso inferior ao do peso de corte deve ser proposta.
- os valores em frequência representam a variável μ dá recorrência mínima necessária,
- dia, mês e ano na subdivisão de frequência correspondem aos períodos t,
- as colunas em qualidade, representam os valores H máximo esperado e L mínimo esperado, para os três períodos t,
- os valores em corte, media é a representação da variável Ξ enquanto qntd a representação da variável β.

Deste modo, as simulações foram executadas conforme as condições apresentadas nessa Seção, e seus resultados serão discutidos a continuação.

5.2. Resultados

A primeira simulação configurou-se com data inicial 02/03/2017(dois de março de dois mil e dezessete) às 06:00(seis horas da manhã). Todas as regras esperadas foram encontradas, a última sendo no dia 24/03/2017 (vinte e quatro de março de dois mil e dezessete) às 05:01 (cinco horas e um minuto da manhã).

Essa simulação contou com 20 diferentes combinações de leituras de sensores e 37 comandos diferentes, totalizando assim 740 combinações distintas.

Durante o período de 21 dias e 23 horas, o *Framework* registrou 812 leituras de sensores e 1590 comandos para equipamentos. A partir desses registros a A.I. gerou 150 possíveis regras para apresentar ao usuário. Dentre essas, 134 foram negadas, três ficaram aguardando a cota e 13 foram aceitas conforme as regras apresentadas na Tabela 3.1. Ao final da execução da simulação, os parâmetros de controle possuíam os valores conforme a Tabela 5.1.

Parâmetros Finais Simulação 1												
cota	**mínimo para apresentação diária**	**frequência**			**qualidade**						**Corte**	
					Dia		**Mês**		**Ano**		**media**	**qntd**
		Dia	**Mês**	**Ano**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**		
5	7	2	22	23	1	10	30	300	360	3600	4,685	1699

Tabela 5.1: Parâmetros Finais Simulação 1 Fonte: Elaborado pelo autor

É possível notar que o valor de “qntd” representa uma quantidade muito maior do que a quantidade de regras existentes. Isso ocorre porque, de acordo com a Fórmula (3.18) a quantidade é atualizada para $\beta_t = (\beta_p + \beta_t)$, assim a quantidade passa a contar diversas vezes a mesma regra, conforme β_t é recalculado.

A segunda simulação configurou-se com data inicial 02/03/2017(dois de março de dois mil e dezessete) às 18:00(seis horas da tarde). Todas as regras esperadas foram encontradas, a última sendo no dia 26/03/2017 (vinte e seis de março de dois mil e dezessete) às 22:00 (dez horas da tarde).

Essa simulação contou com 16 diferentes combinações de leituras de sensores e 27 comandos diferentes. Formando um total de 432 combinações diferentes.

Durante o período de 24 dias e quatro horas, o *Framework* registrou 794 leituras de sensores e 986 comandos para equipamentos. A partir desses registros a inteligência artificial gerou 154 possíveis regras, dentre essas 139 foram negadas, seis ficaram aguardando na lista de espera, uma ficou aguardando a cota e oito foram aceitas conforme as regras apresentadas na Tabela 3.2. Ao final da execução da simulação, os parâmetros de controle possuíam os valores conforme a Tabela 5.2.

Parâmetros Finais Simulação 2												
cota	mínimo para apresentação diária	frequência			qualidade						Corte	
					Dia		Mês		Ano		media	qntd
		Dia	Mês	Ano	Min	Max	Min	Max	Min	Max		
6	7	2	25	28	1	10	30	300	360	3600	4,305	1196

Tabela 5.2: Parâmetros Finais Simulação 2 Fonte: Elaborado pelo autor

Ao contrastar as duas simulações, observa-se alguns pontos interessantes a respeito do funcionamento do sistema, como no gráfico 1 da Figura 5.1 a respeito das entradas de dados recebidos pelo *Framework*, o gráfico 2 da Figura 5.2 a respeito da recorrência de duas regras de cada simulação, e o gráfico 3 da Figura 5.3 sobre a saída produzida pela inteligência artificial.

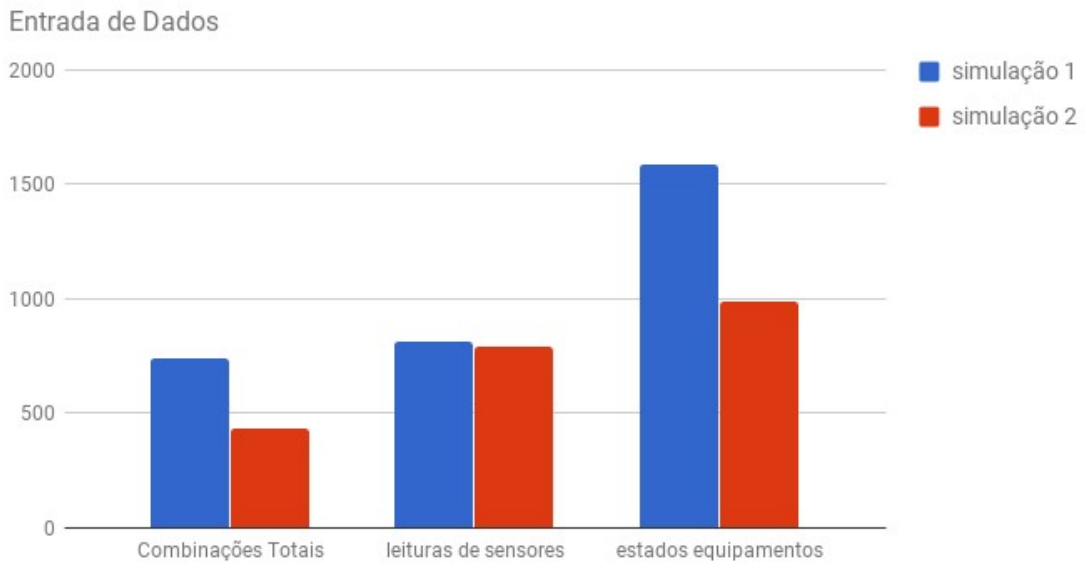


Figura 5.1: Gráfico 1 - Entrada de Dados Fonte: Elaborado pelo autor

Nota-se no gráfico 1 que a quantidade de combinações possíveis entre as leituras de sensores e estados de equipamentos (Combinações Totais) da simulação 1 é bem superior ao da simulação 2. Ou seja, dado o conjunto de leituras de sensores e o conjunto de estados de equipamento, existem 740 combinações de regras distintas para a simulação 1 e 432 para a simulação 2. Impende ressaltar que a quantidade de entrada de leituras de sensores no *Framework* foi próxima nas duas simulações, entretanto a quantidade de leituras de entradas

de estados da simulação 1 foi bem superior ao da simulação 2, sendo 1590 da primeira contra 986, mesmo a segunda simulação tendo sido executada por 53 horas a mais que a primeira.

Outra comparação feita entre as duas simulações, apresentada no gráfico 2, refere-se à primeira regra encontrada de cada simulação (regra 1) e à última regra encontrada (regra 2). Essas regras são a percentagem de aparições da tupla ação em relação a quantidade de leituras de ação e a percentagem de aparições da tupla comando em relação a quantidade de mudanças de estados de equipamentos registrados.

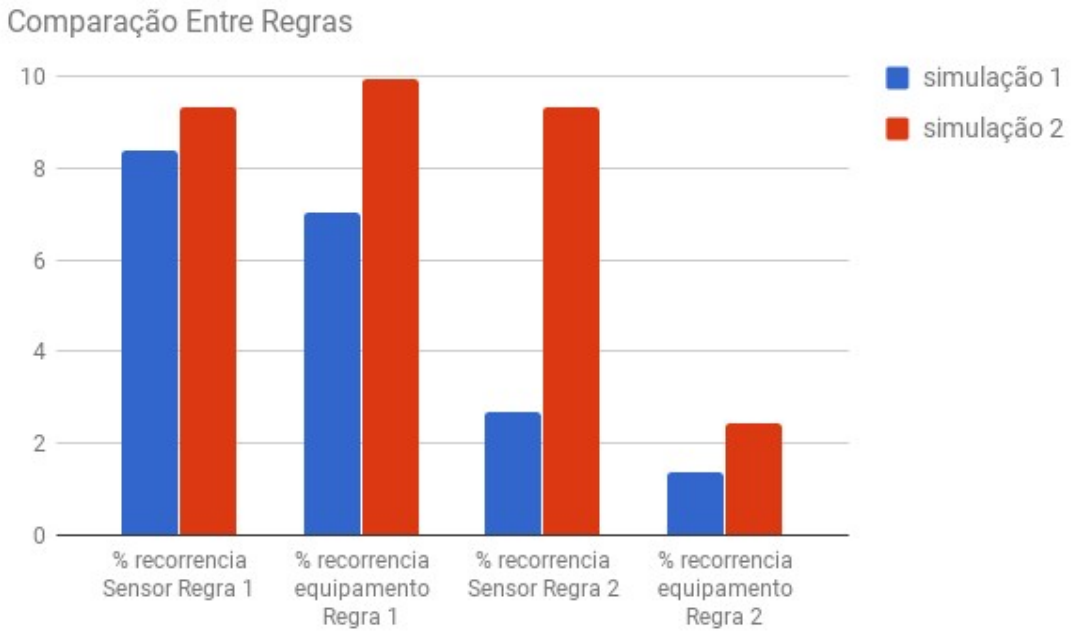


Figura 5.2: Gráfico 2 - Comparação de Regras Fonte: Elaborado pelo autor

Observa-se no gráfico 2 uma diferença considerável entre a simulação 1 a simulação 2. Tanto para a primeira regra encontrada quanto para última regra encontrada, a percentagem da simulação 2 é superior à da simulação 1. Essa percentagem considera a quantidade de vezes que a leitura de sensor ou estado do equipamento aparecem em relação a quantidade total de entradas daquele tipo.

Considerando a variedade de possíveis regras para cada simulação e a percentagem de aparição da menor regra, supõe-se que a última regra da simulação 2 seria encontrada mais rápido que a última regra da simulação 1. Entretanto o tempo de execução da simulação 1 foi menor que o tempo de execução da simulação 2, mesmo a primeira tendo que encontrar mais regras, assim como possuir maior variedade de possíveis regras e sua última regra tendo menor relevância no que tange a quantidade de vezes que ela ocorre.

Além do tempo de execução, a simulação 2 precisou gerar mais possíveis regras que a simulação 1 para encontrar as regras esperadas. Isso é visto no gráfico 3 que apresenta as saídas de dados geradas pela inteligência artificial.

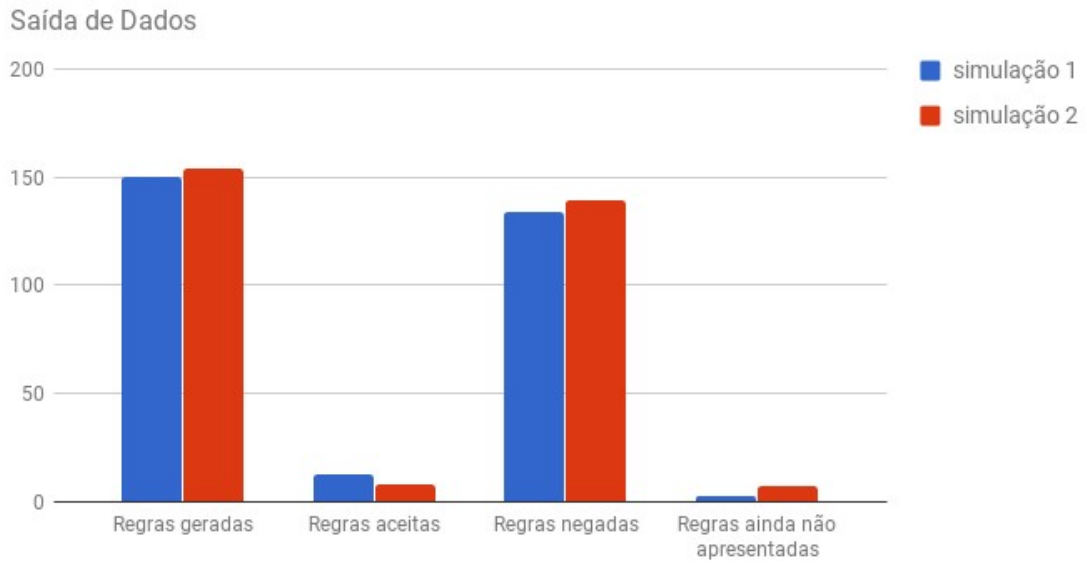


Figura 5.3: Gráfico 3 - Saída de Dados Fonte: Elaborado pelo autor

As quantidades de regras geradas nas duas simulações foram muito próximas. Todavia as quantidades de regras geradas pela quantidade de regras esperadas (regras aceitas) diferenciam-se significante, sendo 19,25 regras geradas por regras aceitas na simulação 2 contra 11,53 na simulação 1.

Enquanto na primeira simulação era necessário encontrar 13 regras dentre 740 possibilidades diferentes, na segunda simulação precisava-se encontrar oito regras dentre 432. Nesse aspecto, a primeira simulação possui um cenário menos propício que a segunda. Isso se contrapõe com o resultado da primeira simulação aparecendo mais rápido e com menor percentagem de erro. Atribui-se essa diferença de resultados ao maior volume de entrada de dados na primeira simulação em relação a segunda, como visto no gráfico 1. O maior volume de dados proporcionou que a A.I. tivesse um aprendizado mais rápido no primeiro caso que no segundo.

Conclui-se pela análise dos resultados dois pontos importantes. O primeiro é que a inteligência artificial utilizada cumpriu seu objetivo nas duas simulações, uma vez que foi capaz de encontrar todas as regras que eram esperadas que fossem encontradas. O segundo é que, haja visto as comparações feitas nessa Seção, a entrada de dados é de tal importância para o aprendizado da inteligência artificial que é capaz de reverter um cenário menos propício e trazer resultados de forma mais eficaz e eficiente.

6. CONCLUSÃO E TRABALHOS FUTUROS

O uso de sistemas domóticos ganha mais destaque a cada dia, principalmente pelo grande ganho em comodidade e economia gerado por seu uso. Da mesma forma os avanços da inteligência artificial têm recebido bastante destaque e sua aplicação tem alcançado diversas áreas distintas. Assim, a associação dessas duas tecnologias tem o potencial de aprimorar a forma como nossas residências são controladas, implicando em aumento de conforto, segurança e economia. Sobre essa ótica, o presente trabalho propôs um sistema domótico integrado com inteligência artificial, capaz de analisar a rotina dos usuários e identificar padrões, transformando-os em regras de automação.

O sistema apresentado por este trabalho foi dividido em duas partes principais. Primeiro, o sistema domótico denominado *Framework* AIHC, construído de forma modular para permitir maior flexibilidade em alterações futuras. Segundo, o módulo de inteligência artificial capaz de trabalhar em conjunto com esse *Framework*.

Analisando os resultados obtidos nas simulações feitas no presente trabalho, percebe-se que o conjunto construído para esse sistema apresentou resultados satisfatórios. O módulo de inteligência artificial desenvolvido foi capaz de cumprir as expectativas. Também é possível ressaltar que resultados consideravelmente melhores foram apresentados para cenários de maior volume de entrada de dados, como se espera de um agente inteligente com aprendizado à medida que sua base de conhecimento aumenta. Com isso, foi possível demonstrar seu ganho de aprendizado de acordo com o volume de dados recebidos como entrada.

Impende ressaltar que se alcançou os objetivos esperados, uma vez que o *Framework* foi desenvolvido de forma modular, possibilitando que futuras melhorias sejam adicionadas de forma simples. Através do ambiente virtual e agente de teste validou-se o modelo proposto, tanto para o *Framework*, quanto para o módulo de inteligência artificial. Nesse cenário o agente inteligente cumpriu a função de encontrar padrões e propor regras de automação.

Afirma-se então que o objetivo geral proposto foi devidamente alcançado. O sistema aqui apresentado é adequado para ser utilizado em ambientes residenciais, controlando seus equipamentos e colaborando para que o(s) morador(es) tenham maior comodidade e segurança. Cabe destacar o fato de ser um sistema livre e de código aberto, o que facilita a difusão do seu uso.

Como possíveis melhoria, é possível citar a incorporação de um módulo de comando por voz. A transcrição de voz para texto já é uma realidade presente em diversos dispositivos, tais como os sistemas operacionais *windows* e *apple,* por meio das Cortana e Siri, respectivamente. Supondo-se a interação multimodal como uma realidade para o controle de todos os dispositivos em um futuro próximo, o módulo de comandos de voz agregaria ainda

mais valor ao sistema. Atualmente o sistema já conta com um módulo de interação com o usuário que aceita comando de textos, dessa forma o sistema já está apto a receber essa melhoria. Um sistema de transcrição de voz para texto pode ser associado ao sistema de comando de texto já existente sem necessidade de alterações no resto do sistema.

Outra melhoria que poderia ser adicionada ao sistema seria um módulo de controle de sinais vitais com um sistema de alerta. Assim o sistema auxiliaria em cuidados para idosos e pessoas com saúde debilitada, enviando alertas para uma equipe médica ou responsável sempre que a condição da pessoa monitorada exigir. Conceitualmente é uma contribuição muito interessante para as pessoas com necessidades de cuidados especiais, haja visto a possibilidade de monitoramento 24 horas por dia.

Esse estudo sumariza-se como a modelagem e implementação bem sucedida de um sistema que permite gerenciar a casa com auxílio de uma inteligência artificial. Sendo um sistema modular, está preparado para receber melhorias conforme a tecnologia avance, facilitando assim que permaneça sempre atualizado e seguindo as tendências e avanços tecnológicos.

BIBLIOGRAFIA

COSTA, L. R.; OBELHEIRO, R. R.; FRAGA, Joni S.. Introdução à Biometria. In: SIMPÓSIO BRASILEIRO DE SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS, 6., 2006, Porto Alegre. **Livro texto dos Minicursos.** Porto Alegre: Sbc, 2006. v. 1, p. 103 - 151.

DATE, C. J.. **Introdução a Sistema de Banco de Dados.** 8. ed. Rio de Janeiro: Campus, 2003.

DEITEL, Paul J.; DEITEL, Harvey M.. **Como programar em C**. 6. ed. São Paulo: Pearson Education - Br, 2011. 828 p. Tradução de: Daniel Vieira.

DIAS, Mateus P.; CARVALHO, José O.f.. A Visualização da Informação e a sua contribuição para a Ciência da Informação. **Datagramazero,** Rio de Janeiro, v. 8, n. 5, p.1-16, out. 2007.

DORIGO, Marco; STÜTZLE, Thomas. **Ant Colony Optimization.** Londres: A Bradford Book, 2004. 319 p. (ISBN-10: 0262042193).

ELMASRI, Ramez; NAVATHE, Shamkant B.. **Sistemas de Banco de Dados.** 6. ed. São Paulo: Pearson Education - Br, 2011. 808 p. Tradução de: Addison Wesley.

ECMA INTERNATIONAL. **ECMA-262**: ECMAScript. 7 ed. Genebra: Ecmascript®, 2016. 39 p.

FRANKLIN, Stan; GRAESSER, Art. Is It an agent, or just a program?: A taxonomy for autonomous agents. In: INTELLIGENT AGENTS III AGENT THEORIES, ARCHITECTURES, AND LANGUAGES, 3., 1996, Budapest. **Proceedings of the Third International Workshop on Agent Theories, Architecture, and Languages.** Berlin: Springer-verlag, 1997. v. 1193, p. 21 - 35. Disponível em: <http://www.inf.ufrgs.br/~alvares/CMP124SMA/IsItAnAgentOrJustAProgram.pdf>. Acesso em: 30 jun. 2017.

KASABOV, Nikola K.. **Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering.** Londres: Bradford, 1998.

MATRIX. [s.l.]: Warner Bros, 1999. (136 min.), son., color.

MCGLINN, Kris et al. SimCon: A Tool to Support Rapid Evaluation of Smart Building Application Design using Context Simulation and Virtual Reality. **Journal Of Universal Computer Science.** Graz, p. 1992-2018. ago. 2010. Disponível em: <http://www.jucs.org/jucs_16_15/simcon_a_tool_to/jucs_16_15_1992_2018_mcglinn.pdf>. Acesso em: 10 mar. 2015.

MEIGUINS, Bianchi Serique et al. Realidade Virtual e Aumentada em Visualização de Informação. In: TORI, Romero; KIRNER, Claudio; SISCOUTTO, Robson. **Fundamentos e Tecnologia de Realidade Virtual e Aumentada.** Belém: Sbc, 2006. Cap. 21. p. 319-326.

MESSIAS, Alan F.. **Edificios "inteligentes":** A domótica aplicada à realidade Brasileira. 2007. 42 f. TCC (Graduação) - Curso de Engenharia de Controle e Automação, Universidade Federal de Ouro Preto, Ouro Preto, 2007.

OSÓRIO, Fernando; BITTENCOURT, João Ricardo. Sistemas Inteligentes baseados em Redes Neurais Artificiais aplicados ao Processamento de Imagens. In: WORKSHOP DE INTELIGÊNCIA ARTIFICIAL, 1., 2000, Santa Cruz do Sul. **UNISC.** Santa Cruz do Sul: Unisc, 2000. p. 2 - 30.

PRUDENTE, Francesco. **Automação Predial e Residencial:** uma Introdução. São Paulo: Ltc, 2011. 228 p.

RUSSELL, Stuart J.; NORVIG, Peter. **Artificial Intelligence:** A Modern Approach. New Jersey: Prentice Hall, 1995. 905 p. Disponível em: <http://stpk.cs.rtu.lv/sites/all/files/stpk/materiali/MI/Artificial Intelligence A Modern Approach.pdf>. Acesso em: 01 jul. 2017.

SCHILDT, Herbert; GUNTLE, Greg. **Borland C++ Builder:** a referência completa. Rio de Janeiro: Elsevier, 2001. 771 p. Tradução de Edson Furmankiewicz.

SETZER, Valdemar W.. **Banco de dados; conceitos; modelos; gerenciadores; projeto lógico; projeto físico.** 2. ed. São Paulo: Edgard Blucher, 1987. 289 p.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S.. **Sistema de banco de dados.** Rio de Janeiro: Elsevier, 2006. (ISBN-10: 85-352-4535-9). Tradução de Daniel Vieira.

SILVER, Edward A.. An Overview of Heuristic Solution Methods. **The Journal Of The Operational Research Society.** Birmingham, p. 936-956. maio 2004.

SPENCE, Robert. **Information Visualization**.Michigan: Addison-Wesley, 2001. 206p.

STROUSTRUP, Bjarne. Learning Standard C++ as a New Language. **The C/c++ Users Journal.** Londres, p. 43-54. maio 1999. Disponível em: <http://www.stroustrup.com/new_learning.pdf>. Acesso em: 15 maio 2015.

TORI, Romero; KIRNER, Claudio. Fundamentos de Realidade Virtual. In: TORI, Romero; KIRNER, Claudio; SISCOUTTO, Obson. **Fundamentos e Tecnologia de Realidade Virtual e Aumentada:** Realidade Aumentada. Belém: Sbc, 2006. Cap. 1. p. 2-21.

ANEXO I – BANCO DE DADOS CENTRA

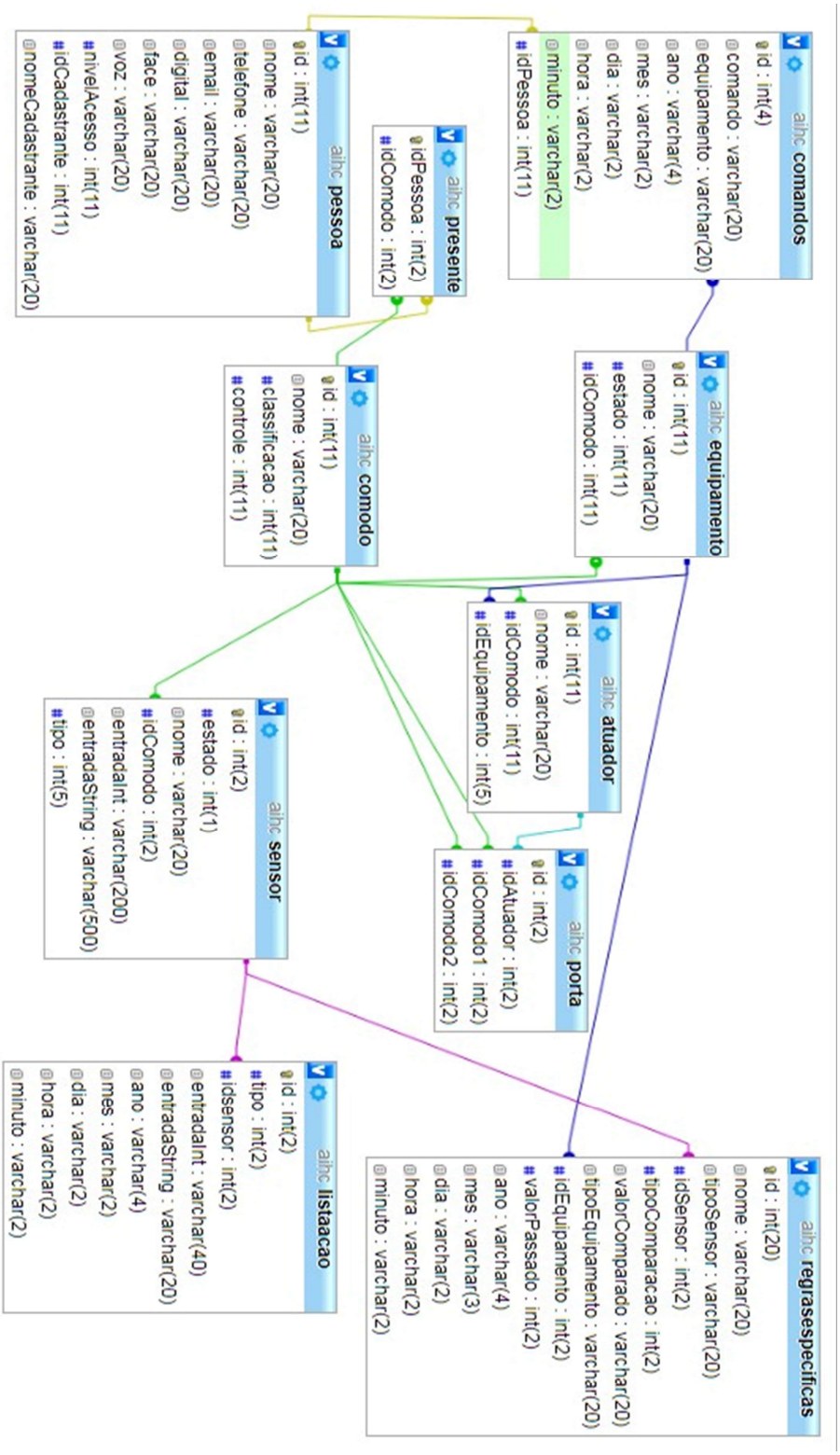


Figura 6.1 Banco de dados Central Completo Fonte: Elaborado pelo autor

Figura 6.2 Banco de Conhecimento Completo Fonte: Elaborado pelo autor

ANEXO II – TABELA DE VARIÁVEIS

Símbolo	Definição	Equação
G	Registro de comando ou ação	(3.1)(3.2)
s	Sensor	(3.1)(3.5)(3.6)(3.7)(3.8)(3.9)(3.11)(3.13)(3.14)(3.16)(3.17)
l	Leitura de sensor	(3.1)(3.5)(3.6)(3.7)(3.8)(3.9)
R	Recorrência	(3.1)(3.2)(3.8)(3.12)
t	Período	(3.1)(3.2)(3.3)(3.5)(3.6)(3.7)(3.8)(3.9)(3.14)(3.15)(3.16) (3.19)(3.20)(3.21)(3.22)
e	Equipamento	(3.2)(3.5)(3.6)(3.7)(3.8)(3.9)(3.11)(3.13)(3.15)(3.16)(3.17)
v	Valor do equipamento	(3.2)(3.5)(3.6)(3.7)(3.8)(3.9)
a	Momento atual	(3.3)(3.7)(3.10)(3.19)
S	Registro salvo no banco	(3.4)
A	Relação entre tupla	(3.5)(3.6)
H	Horas das recorrências	(3.5)(3.6)
i	Id da tupla	(3.5)(3.6)(3.7)(3.8)(3.9)(3.11)
V	Viabilidade da regra	(3.7)(3.14)(3.15)(3.16)
Ω	Regra especifica	(3.7)(3.14)(3.15)(3.16)
Θ	Regra negada	(3.7)
μ	Frequencia minima	(3.7)(3.19)
κ	Varivel contrle AI	(3.7)
δ	Tupla com menor recorrencia	(3.7)(3.8)(3.9)
P	Possivel regra	(3.10)(3.12)
b	Momento da ultima ordenação	(3.10)(3.19)(3.20)
c	Cota variavel de controle	(3.10)(3.19)
VT	Peso de relevancia de regra	(3.11)(3.12)
ρ	Probabilidade de aceitar	(3.11)(3.13)
γ	Distancia geografica	(3.11)(3.17)
TR	Total de regras aceitas	(3.13)(3.14)(3.15)(3.16)
TP	Total de regras propostas	(3.13)(3.14)(3.15)(3.16)
y()	Função existencia	(3.14)(3.15)(3.16)
Ξ	Valor de corte	(3.18)
β	Quantidade de regras	(3.18)
n	Negadas	(3.18)(3.19)(3.20)(3.21)(3.22)
p	Aceitas	(3.18)(3.19)(3.20)(3.21)(3.22)

Símbolo	Definição	Equação
Δ	Diferença de regras	(3.19)(3.20)(3.21)(3.22)
ac	Quantidade de aceitos	(3.20)
ng	Quantidade de negados	(3.20)
Q	Quantidade de proposta	(3.21)(3.22)
H	Máximo esperado	(3.21)(3.22)
L	Mínimo esperado	(3.21)(3.22)

Tabela 6.1: Variáveis das Equações do Capitulo 3 Fonte: Elaborado pelo autor

ANEXO III - AGENTE INTELIGENTE

1. Inicialização do Sistema

Desenvolveu-se uma função com o objetivo de iniciar todos os módulos desse sistema, vista na Figura 7.1. Consiste em três funções para inicialização de módulos seguida de um *loop* infinito. Adicionalmente inseriu-se a função Sleep, que cria uma pausa no sistema entre a inicialização de um módulo e outro, para que não haja sobrecarga do S.O. (Sistema Operacional) durante a inicialização.

```
1  ArtificialIntelligence(){
2    systemStart("cronometro.exe");//inicia o cronometro que marca mudança de periodo
3    Sleep(5000);
4    systemStart("qualidadeTotal.exe");//inicia o sistema que fará o controle de qualidade
5    Sleep(5000);
6    systemStart("controleDeAgentes.exe");//inicia o sistema que controla os periodos
7    while(true){
8      //confere se esta na hora de iniciar um novo agente (para cada periodo)
9      Sleep(5000);
10     acessaKnowledge(BD);
11     if(consultaStart(BD,"dia")){setStart(BD,"dia"); systemStart("agentes_simples.exe","dia");}
12     if(consultaStart(BD,"mes")){setStart(BD,"mes"); systemStart("agentes_simples.exe","mes");}
13     if(consultaStart(BD,"ano")){setStart(BD,"ano"); systemStart("agentes_simples.exe","ano");}
14     fechaCon(BD);
15   }
16   return EXIT_SUCCESS;
17 }
```

Figura 7.1: Aplicação Agente Completo Fonte: Elaborado pelo autor

A função SystemStart possui duas construções, uma para receber um parâmetro e outra para receber dois parâmetros. Ambas iniciam um módulo do sistema. Elas se diferem, pois, alguns módulos precisarem de parâmetros de inicialização, sendo necessários dois parâmetros para a função de inicialização desses.

Os três primeiros módulos apresentados são:

a) O cronometro.exe, responsável por monitorar a data e hora, fazendo a alteração das flag de controle da validade de cada período, conforme esses terminam. Com isso informa aos agentes simples o momento de terminar sua execução.

b) O qualidadeTotal.exe, responsável por monitorar a quantidade de regras propostas, assim como a porcentagem de regras aceitas e negadas dentre elas. Esse módulo será mais detalhado posteriormente.

c) O controleDeAgentes.exe, responsável por inicializar os módulos Gerador de Regras, Ordenador de Regras e o Proponente de Regras. Esses três módulos serão explicados em mais detalhes neste Capítulo.

O *loop* infinito objetiva averiguar a hora de iniciar um novo agente simples para cada período. Conforme as mudanças das *flag's* controladas pelo cronômetro.exe ocorrem, o

agente simples finaliza sua execução e altera essa *flag*. Sinalizando dessa forma a liberação para iniciar outro agente simples para o período em questão.

Logo esse *loop* faz a leitura dessas *flag's* constantemente. Através da função consultaStart() avalia se o sistema está pronto para a inicialização de um novo agente simples para determinado período. Em caso afirmativo, altera-se a *flag* para sinalizar que o agente foi iniciado e inicia o agente para o período especificado.

Uma vez realizadas essas tarefas, o módulo de Inteligência Artificial está totalmente iniciado.

2. Agente Simples

O processo de análise do ambiente feito pelo módulo Agente Simples utiliza-se de três classes. A primeira classe Agente Base (AgenteBase) é a classe base para o desenvolvimento das duas outras, Agente Ação (AgenteAcao) e Agente Comando (AgenteComando). A construção da classe Agente Base é vista na Figura 7.2.

```
class AgenteBase{
  protected:
   id; // id do agente
   ano; // ano da ação/comando
   mes; // mes da ação/comando
   dia; // dia da ação/comando
   hora; //hora da ação/comando
   minuto; // minuto da ação/comando
   flagAceite; // flag que determina se o comando já foi criado
   countTrys; // contador de execuções dessa pre-regra
   countTrysPost; //contador de execuções post-regra criada
  public:
  AgenteBase(); // construtor do agente
  ~AgenteBase(); // destrutor do agente
  AgenteBase( id, ano, mes, dia, hhora, min, flagAceite, countTrys, countTrysPost);//construtor instanciado
};
```

Figura 7.2: Agente Base Fonte: Elaborado pelo autor

O Agente Base é uma classe contendo atributos compartilhados pela Agente Comando e Agente Ação. Foram aplicados conceitos de herança para implementação das duas últimas.

As variáveis associadas a classe base são i) um id da primeira aparição no período; ii) cinco variáveis para armazenar a data e hora da primeira aparição do registro; iii) uma variável que contém uma *flag* para saber se já existe um registro equivalente detectado como recorrente na tabela equivalente; iv) uma variável para armazenar quantas recorrências essa ação/comando teve no período; e v) uma variável que armazena a contagem de recorrências após ter sido criado uma regra derivada desse registro.

Os métodos criados para essa classe são um construtor vazio, cuja utilidade é alocar espaço para um objeto, um destrutor para liberar espaço de memória alocado para um objeto que não será mais usado e um construtor que recebe os valores para instanciar todas as variáveis da classe para um determinado objeto.

As duas classes que herdam de Agente Base possuem funções semelhantes. Enquanto Agente Ação é a responsável por procurar recorrência em leituras de sensores, Agente Comando tem a função de procurar a recorrência das alterações de estados dos equipamentos. O primeiro utiliza a Fórmula (3.1) enquanto o segundo a Fórmula (3.2).

Um exemplo de comando é acender uma luz (altera estado da lâmpada de 0 para 1), um exemplo de ação é a leitura positiva de presença de determinado usuário por um determinado sensor (leitura do sensor identificação de usuário x).

A classe Agente Simples (AgenteSimples) foi desenvolvida com o objetivo de controlar um objeto Agente Comando e um objeto Agente Ação vinculados a determinado período (dia, mês, ano, etc…). Ou seja, existe um Agente Simples para cada período definido, e cada um deles controla os dois agentes citados para que façam a busca referente a esse período definido.

O Agente Simples é responsável também por monitorar a *flag* de fim de período controlada pelo cronometro.exe já citado. Quando essa *flag* é alterada para fim de período, os dois agentes controlados por essa classe são finalizados. Após essa finalização altera-se a *flag* informando ao ArtificialIntelligence() para iniciar um novo agente para aquele período. Para melhor entendimento a Figura a seguir demonstra os atributos da classe Agente Simples.

```
1  class AgenteSimples{
2    private:
3    tempoAgente; // tempo de atuação do agente
4    public:
5    ~AgenteSimples();
6    AgenteSimples(periodo);
7  };
```

Figura 7.3: Agente Simples Fonte: Elaborado pelo autor

A classe Agente Simples foi implementada com um atributo, que possui o objetivo de determinar qual o período de atuação do agente. Possui um construtor que recebe o período ao qual esse agente deve atuar, dentro desse construtor o agente cria um objeto da classe Agente Ação e um objeto da classe Agente Comando. Cada um desses objetos recebe o período, a data e a hora atual.

As classes Agente Ação e a classe Agente Comando são bastante semelhantes em suas funções. A primeira busca as leituras percebidas pelos sensores (ação) e os agrupa em uma lista, a segunda busca mudanças de estados de equipamentos (comandos) para agrupá-los em uma lista. Ou seja, enquanto um compara semelhanças entre as ações (Agente Ação) o outro compara as semelhanças entre os comandos (Agente Comando).

A classe Agente Ação possui 4 atributos, sendo: um para o id do sensor, um para o tipo de sensor, um para a entrada lida no sensor e um para apontar para um próximo objeto

dessa classe. Dessa forma, cada objeto é responsável por armazenar a primeira ocorrência da tupla (leitura,sensor) no período e para cada ocorrência seguinte dessa tupla o valor de countTrys (atributo da classe base) é incrementado em um. A Figura 7.4 apresenta a implementação dessa classe.

```
class AgenteAcao: public AgenteBase{
  protected:
   idSensor; // id do sensor que disparou a ação
   tipo; //tipo de sensor
   entrada; // entrada lida pelo sensor
  public:
  AgenteAcao *prox; // ponteiro para a proxima ação registrada pelo agente
  ~AgenteAcao(); // destrutor do objeto
  // construtor instanciado
  AgenteAcao(tipo,idSensor,entrada,id, ano, mes, dia, hora,  minuto, flagAceite, countTrys, countTrysPost);
  //inicia agente pela chamada do agente simples
  AgenteAcao *criaAcaoAgenteSimples( periodo,  ano,  mes,  dia, hora, minuto);
  //atua durante o periodo ativo lendo os registros da listaacao
  AgenteAcao *criaAcao( condicao, periodo);
  //busca dados da ação
  void addAcaoId( periodo, idOriginal);
  //adiciona novo nó na lista usando o addLstCmd ou incrementa countTrys caso exista recorrencia anterior para essa acao
  void addAcao( periodo, tipo, idSensor, entrada, id, ano, mes, dia, hora,  minuto, flagAceite, countTrys, countTrysPost);
  //função de busca de ações iguais/semelhantes
  AgenteAcao *buscaAcao( tipo, idSensor, entrada);
  //add o ponteiro no final da lista
  void addLstAcao(AgenteListaAcao *novo);
};
```

Figura 7.4: Classe Agente Acao Fonte: Elaborado pelo autor

A classe Agente Comando é análoga a classe supracitada, excetuando seus atributos, que são: comando, utilizado para salvar o valor do estado; equipamento que salva o id do equipamento; o idPessoa que armazena o id da pessoa que enviou o comando para alterar o estado do equipamento; e um ponteiro que aponta para o próximo objeto dessa classe. Portanto, cada objeto funciona da mesma forma que os da classe Agente Ação, mas enquanto um agrupa as tuplas (leitura,sensor) o outro agrupa as tuplas (estado,equipamento,pessoa). A implementação da classe Agente Comando é apresentada na Figura 7.5.

```
class AgenteComando: public AgenteBase{
  protected:
    comando;// comando enviado
    equipamento; // equipamento(s) envolvido
    idPessoa; // identificador da pessoa que enviou o comando
  public:
  AgenteComando *prox; // ponteiro para o proximo comando a ser armazenado
  ~AgenteComando(); // destrutor
  //construtor instanciado
  AgenteComando(comando,equipamento,idPessoa,id,ano,mes,dia,hora,minuto,flagAceite,countTrys,countTrysPost);
  //inicia agente pela chamada do agente simples
  AgenteComando *criaCmndAgenteSimples(periodo,ano,mes,dia,hora,minuto);
  //atua durante o periodo ativo lendo os registros da tabela comandos
  AgenteComando *criaCmd(condicao,periodo);
  //busca dados do comando
  void addCmdId(periodo,idOrigianl);
  //adiciona novo nó na lista usando o addLstCmd ou incrementa countTrys caso exista recorrencia anterior para esse comando
  void addCmd(periodo,comando,equipamento,idPessoa,id,ano,mes,dia,hora,minuto,flagAceite,countTrys,countTrysPost);
  //função de busca comando igual/semelhante
  AgenteComando *buscaCmd(cmd,equipamento,idPessoa);
  //cria um novo no a ser add na lista
  void addLstCmd(AgenteComando *novo);
};
```

Figura 7.5: Classe Agente Comando Fonte: Elaborado pelo autor

Os métodos implementados para as duas classes também são análogos, sendo 8 em cada, os quais possuem as seguintes funções:

i) destrutor (~AgenteAcao() e ~AgenteComando()), desaloca espaço de memória alocado para o objeto;

ii) construtor (~AgenteAcao() e ~AgenteComando()), recebe os valores da tupla e os valores para preencher os atributos herdados da classe Agente Base;

iii) cria lista para agente simples (criaAcaoAgenteSimples() e criaCmdAgenteSimples()), essa função é chamada pelo agente simples recebendo como parâmetro o período e a data e hora atual. Após iniciado cria-se um ponteiro que servirá como lista de objetos dessa classe assim como a condição que será usada na consulta SQL segundo o período do agente e a data informada. Em seguida chama a função do item iv passando a condição e o período;

iv) cria lista com condição (criaAcao() e criaCmd()), essa função fica ativa enquanto o período for válido. Enquanto ativa, busca todos os registros ainda não lidos pelo agente (período) e para cada novo registro encontrado chama a função do item vi, passando como parâmetro o período e o id do registro;

v) add agente id (addAcaoId() e addCmdId()), acessa o registro referente ao id recebido para buscar os valores equivalentes aos atributos da classe(a tupla de valores, assim como data e hora);

vi) add agente (addAcao() e addCmd()), passa a tupla de valores recebidos para a função descrita em vii. Sendo o retorno da função um ponteiro válido, incrementa o countTrys do objeto nesse ponteiro em um e compara o novo valor de countTrys com o mínimo de frequência. Se a frequência for igual ou maior que a esperada, é criado um registro na tabela de conhecimento referente ao agente em questão. Caso o retorno de vii seja um ponteiro nulo, instancia um objeto usando ii e passando todos parâmetros recebidos de iv; e chama o método em viii passando o ponteiro do objeto criado;

vii) busca agente (buscaAcao() e buscaCmd()), percorre a lista de objetos procurando a tupla de valores recebidos nos nós dessa lista. Retornando seu endereço caso encontre ou um ponteiro nulo caso contrário;

viii) add lista (addLstAcao() e addLstCmd()), recebe um ponteiro para um objeto e adiciona no final da lista.

3. Gerador de Regras

O gerador de regras busca nas tabelas de recorrências os registros que foram armazenados pelos agentes descritos na subseção anterior. Seu objetivo é criar agrupamentos de tuplas (ação,comando). O método utilizado para definir o agrupamento dessas tuplas divide-se em duas partes a) o intervalo de tempo entre a ação e o comando, e b) o resultado do teste que averigua se já existe uma regra equivalente e quantas vezes essa regra foi negada pelo usuário. A lógica apresentada na Figura 7.6 implementa as partes (a) e (b) supracitadas.

```
build_Proposta(string periodo){
  while(true){
    acessaBD(BD1);
    acessaKnowledge(BD2);
    nextId = buscaProximoId(BD2,periodo);
    lista2 = buscaListaAcao(BD2,periodo);
    lista1 = buscaListaComando(BD2,periodo);
      while(existeidListaVetor(lista1)){
        buffer = lista2;
        tentativas = numeroTentativas(DB2,nextId);
        minimoTentativas = consultaFrequencia(periodo);
        sleep(300);
        if(tentativas >= minimoTentativas){
          while(existeidListaVetor(buffer){
            nextId2 = idDaListaVetor(buffer);
            tentativas2 = numeroTentativas(DB2,nextId);
            if(tentativas2 >= minimoTentativas){
              if(comparaCompatibilidade(nextId,nextId2) && podeCriarRegra(nextId,nextId2)){
                criaRegraPossivel(periodo,nextId,nextId2);
                alteraQauntidadePropostasFeitas(periodo);
              }
            }
            buffer = novaListaVetor(buffer);
          }
          }
        lista1 = novaListaVetor(lista1);
        atualizaProximoId(periodo,nextId);
      }
    zeraProximoId(periodo);
  }
}
```

Figura 7.6: Cria possível regra Fonte: Elaborada pelo autor

A função build_Proposta() é iniciada recebendo um período, que determina qual o intervalo de atuação. Em seguida inicia um *loop* infinito. Onde repete-se quatro partes:

i) é feita a inicialização das conexões com os bancos de dados e as listas de id's nas tabelas disponibilizadas pelo agente simples;

ii) é iniciado um loop que faz uma varredura para cada comando existente na lista de comandos. Verificando se o número de recorrência desses ainda está dentro do mínimo exigido pelo controle de frequência. Essa comparação é feita pelo fato da frequência mínima ser variável, tal que ocasionalmente um registro que antes atendia o requisito mínimo deixe de atende-lo;

iii) para cada comando validado em ii, inicia-se um *loop* que percorre as ações de forma análoga ao que foi feito para os comandos. Para cada ação validada cria-se a tupla (ação,comando) e aplica-se às Fórmulas (3.5) e (3.6). Caso seja satisfeita essas condições é aplicada a Fórmula (3.7). Essa última condição sendo satisfeita, é criado um registro de possível regra.

iv) ao terminar a leitura de todos registros disponíveis nas duas listas, a última função do *loop* é voltar o id que será recuperado por buscaProximoId() para o primeiro registro existente. Por consequência quando o loop infinito recomeça todo processo é feito novamente.

Sumariza-se a função comparaCompatibilidade() como a implementação das Fórmulas (3.5) e (3.6), enquanto podeCriarRegra() tem o intuito de reproduzir a Fórmula (3.7).

Essa última também usa um mecanismo de controle para não permitir que regras criadas aguardando ser apresentadas ao usuário sejam criadas novamente, evitando com isso redundância de propostas.

4. Ordenador de Regras

O ordenador de regras é um dos módulos mais complexos da inteligência artificial, como nota-se na descrição de sua modelagem apresentada previamente. Para sua implementação foram necessárias diversas funções que realizam as comparações apresentadas nas Fórmulas da Seção 3.3.3.

O módulo possui um loop infinito que chama a função ordenaRegras() e após a execução dessa faz uma pausa no sistema. Após a pausa o processo é repetido. A função ordenaRegras() é quem efetivamente define a ordenação das regras a se apresentar ao usuário. A lógica utilizada para sua implementação é vista na Figura 7.7.

```
1  ordenaRegras(){
2    acessaKnowledge(BD);
3    id = totalPossivelRegras(DB);
4    ultimoId = ultimoIdContrle(BD);
5    cota = cotaLista(DB);
6
7    if(id > ultimoId + cota){
8      nextId = ultimoId+1;
9      fila = new filaRegras();
10     while(id >= nextid ){
11     proximo = new ordenadorDeRegra(nextId);
12     fila.addNovo(proximo);
13     nextId++;
14     }
15     espera = listaIdsEspera(BD);
16     while(existeIdLista(espera)){
17     nextIdEspera = idDaListaVetor(espera)
18     proximo = new ordenadorDeRegra(nextIdEspera);
19     fila.addNovo(proximo);
20     espera = novaListaVetor(espera);
21     }
22     apagaRegistrosEspera(BD);
23     filtraLista(BD,fila);
24     atualizaUltimoIdControle(BD,nextId);
25   }else
26   if(faltaMinimoApresentar(BD)){
27       manipulaListaEspera(BD);
28   }
29 }
```

Figura 7.7: Ordena Regras Fonte: Elaborada pelo autor

A função inicia abrindo uma conexão com o banco de conhecimento e carregando as variáveis id, ultimoId e cota que representam respectivamente a) o número de registro na tabela de possível regras, b) o id do último registro ponderado e c) o valor da variável cota. Essa última é um dos parâmetros de controle do sistema.

Após carregar essas variáveis é feito um teste condicional (linha 7) que verifica se após a última ordenação (b) o sistema já recebeu registros de possíveis regras (a) suficientes para satisfazer a cota (c) estipulada. Esse teste condicional é a implementação da Fórmula (3.10). Caso a condição da linha 7 seja atendida seguem-se três passos (i,ii,iii), caso a condição não seja satisfeita executa-se o passo (iv). Os quatro passos citados são:

i) cria um objeto "fila" da classe Fila Regras Ordenadas (filaRegrasOrdenadas). O *loop* da linha 10 atua criando um objeto da classe Ordenador de Regra (ordenadorDeRegra), que pondera cada registro na tabela de possível regra ainda não ponderado por essa função. Cada objeto criado é adicionado na fila pelo método addNovo() da classe Fila Regras Ordenadas.

ii) recebe uma lista com o id de todos registros na fila de espera e para cada um desses registros cria um objeto da classe Ordenador de Regras que em seguida é adicionado na lista através do método addNovo().

iii) apaga os registros na tabela fila de espera, uma vez que esses foram adicionados no objeto "fila". Chama a função filtraLista() que define dentre os objetos da "fila" quais regras devem ser propostas e quais serão armazenadas na tabela fila de espera. Após a execução da função filtraLista() altera o valor do último id lido (a) na linha 24.

iv) a função testa se o mínimo de propostas diárias foi feito ao usuário. Caso não tenha sido, a função manipulaListaEspera() é chamada. Essa função busca as regras de maior peso na lista de espera para que sejam propostas ao usuário no intuito de cumprir a meta mínima de propostas diárias.

A classe Fila de Regras (filaRegras) utilizada na função ordenaRegras() têm sua estrutura vista na Figura 7.8. Essa classe possui uma variável que aponta para um objeto da classe Ordenador de Regra e uma variável que aponta para outro objeto da classe Fila de Regras.

```
class FilaRegras{
public:
  ordenadorDeRegra *no;
  FilaRegras *prox;
  FilaRegras(ordenadorDeRegra *head){
        this->no = head;
        this->prox = NULL;
  };
  ~FilaRegras(){
    if(this->prox != NULL){
    (this->prox)->~FilaRegras();
    }
    delete no;
    delete prox;
  };
  FilaRegras *ordenadaF(FilaRegras *head,FilaRegras *novo){
    if(((novo->no)->relevancia) > ((head->no)->relevancia)){
     (novo->prox) = head;
     return novo;
    }
    if((head->prox) != NULL){
    head->prox = ordenadaF((head->prox),novo);
    return head;
    }else (head->prox)=novo;
    return head;
  }
};
```

Figura 7.8: Classe Fila de Regras Fonte: Elaborada pelo autor

A classe da Figura 7.8 possui três métodos: i) um destrutor para liberar o espaço de memória alocado para a fila, ii) um construtor que recebe um ponteiro para um objeto do tipo Ordenador de Regras. Esse ponteiro é passado para o atributo da classe (*no), e iii) um método que ordena a fila conforme o valor do peso de cada nó.

Esse último método recebe como parâmetro um ponteiro para o primeiro objeto da fila atual e uma fila contendo um objeto. Em seguida compara o valor do peso do novo nó (*novo) com os nós já existentes na fila (*head), começando pelo primeiro. Quando a comparação encontra um nó que tenha valor menor ou igual a do novo nó, o novo é adicionado nesse ponto. Caso ele seja menor que todos os nós existentes, ele é adicionado ao final da fila. Dessa forma cria-se uma fila ordenada de forma decrescente em relação ao peso da possível regra.

A classe Ordenador de Regras é usada para ponderar e armazenar os valores da possível regra. Seus atributos e os seus dois métodos são visto na na Figura 7.9.

```
class ordenadorDeRegra{
  public:
     id;
     idEq;
     idS;
     idComodoS;
     idComodoE;
     relevancia;
  ordenadorDeRegra(key){
   id =key;
   acessaBanco(BD);
   acessaKnowledge(BD2);
   idEq = consultaEquipamentoPossivelRegra(BD2,id);
   idS = consultaSensorPossivelRegra(BD2,id);
   idComodoS = consultaComodoSensor(BD,idS);
   idComodoE = consultaComodoEquipamento(BD,idEq);
   fechaCon(BD2);
   fechaCon(BD);
   Sleep(100);
   relevancia = calculaRelevancia(idComodoE,idComodoS,idEq,idS);
  };
  ~ordenadorDeRegra(){};
};
```

Figura 7.9: Classe Ordenador de Regras Fonte: Elaborado pelo autor

Os seis atributos dessa classe são: i) id, representa o id da possível regra sendo ponderada, ii) idEq, representa o id do equipamento presente nessa regra, iii) idS, representa o id do sensor vinculado a essa regra, iv) idComodoS, representa o id do cômodo no qual o sensor em iii está alocado, v) idComodoE, é o mesmo que iv mas para o equipamento e vi) relevância, é o valor que será ponderado para a possível regra da tupla (equipamento,sensor).

Os dois métodos são: um destrutor e um construtor. O construtor dessa classe recebe a variável key que é proveniente do gerador de regras, e a armazena no atributo i. Esse id é utilizado para mediante uma consulta ao banco recuperar os valores dos atributos ii,iii,iv e v.

Para a ponderação do valor do atributo relevância é utilizada a função calculaRelevancia() a qual a lógica é apresentada na Figura 7.10. Para esse cálculo é levado em conta os quatro atributos recuperados do banco que são os parâmetros passados para a função.

```
1  calculaRelevancia(idComodoEquipamento,idComodoSensor,idE,idS){
2    acessaBD(BD);
3    acessaKnowledge(BD2);
4    //recebe o valor especifico deacordo com distancia geografica
5    vizinhos = vizinhoComodo(BD,idComodoEquipamento,idComodoSensor);
6    //define qual o peso  distancia entre (equipamento,sensor)
7    switch(vizinhos){
8      case 1 : peso = 1; break;
9      case 2 : peso =0.8; break;
10     case 3 : peso =0.4; break;
11     default : peso =0.1; break;
12   }
13   aceitoE = totalRegrasAceitas(BD,idE); //quantidade de regras aceitas para equipamento
14   aceitoS = totalRegrasAceitas(BD,idS); //quantidade de regras aceitas para sensor
15   negadoE = totalRegrasNegadas(BD2,idE); //quantidade de regras negadas para equipamento
16   negadoS = totalRegrasNegadas(BD2,idS); //quantidade de regras negadas para sensor
17   ligacao = totalRegrasAceitas(BD,idE,idS); //quantidade de regras aceitas para tupla
18   ligacaoN = totalRegrasNegadas(BD2,idE,idS); //quantidade de regras negadas para tupla
19   fechaCon(BD2);
20   fechaCon(BD);
21   // calcula a % de regras aceitas para aquele equipamento
22   valorE = aplicaFormula(aceitoE,negadoE);
23   // calcula a % de regras aceitas para aquele sensor
24   valorS = aplicaFormula(aceitoS,negadoS);
25   // calcula a % de regras aceitas para a tupla (equipamento + sensor)
26   valorD = aplicaFormula(ligacao,ligacaoN);
27   // aplicação dos pesos de acordo com cada probabilidade * pelo peso da distancia
28   valorT = ((valorE*2) + (valorS*3) + (valorD*5)) * peso;
29   Sleep(10);
30   return valorT;
31 }
```

Figura 7.10: Calcula Relevância Fonte: Elaborado pelo autor

Calcula Relevância implementa a Fórmula (3.11) para ponderar o valor da regra em questão. A primeira coisa feita é criar dois acessos, um ao banco de conhecimento e outro ao banco central.

Após criado o acesso, é usada a função vizinhoComodo() que retorna: um, caso a tupla esteja no mesmo cômodo; dois, caso sejam vizinhos diretos; três, caso estejam separados por apenas um cômodo; e quatro, em qualquer outro caso.

A linha 7 apresenta um comando condicional que utiliza do valor retornado pela função vizinhoComodo() para implementar a Fórmula (3.17) que pondera a regra segundo a distância geográfica entre o equipamento e o sensor.

Na sequência a função carrega seis variáveis. Sendo três correspondendo a regras aceitas (a) e três a regras negadas (b), das quais: um par (a,b) para equipamento, um par (a,b) para sensor e um par (a,b) para a tupla (equipamento,sensor). Ou seja, essa recuperação de dados correspondem as Fórmulas (3.14), (3.15) e (3.16). Em seguida o acesso ao banco é encerrado.

Com os valores das variáveis carregadas, são feitas três chamadas a função aplicaFormula(), uma para o par do equipamento, uma para o par do sensor e uma para o par da tupla (sensor,equipamento). Essa função é vista na Figura 7.11, e refere-se a implementação da Fórmula (3.13).

```
1  aplicaFormula(aceito,negado){
2    // calcula a % de regras aceitas para aquele equipamento
3    if(aceito){return (aceito/(aceito + negado));}
4     else{
5      if(!negado){ return 0.8;}
6        else return (0.8/negado);
7     }
8  }
```

Figura 7.11: Calcula Probabilidade Fonte: Elaborado pelo autor

Entende-se a Figura 7.11 como: caso exista algum registro aceito para o item em questão, aplica-se a fórmula na linha 3. Caso não exista registro aceito e registro negado a função retorna oito décimos, caso exista apenas registro negado, a função retorna a divisão de oito décimos pela quantidade de registros negados.

O retorno das três chamadas da função aplicaFormula() possibilitam o cálculo da Fórmula (3.11) esse resultado é o retorno da função calculaRelevancia() sendo armazenado na variável relevância do objeto da classe Ordenador de Regras que fez a chamada.

Com isso chega-se a função filtraLista() vista na Figura 7.12. E como já dito, seu objetivo é decidir qual regra será proposta e qual será enviada para a lista de espera.

```
1  filtraLista(BD,lista){
2    while(lista != NULL){
3      peso = (lista->no)->relevancia; //pega o peso do lista nó da fila
4      corte = consultaCorte(BD);//consulta valor de corte para regras
5      faltando = faltaMinAP(BD); // consulta quantas regras faltam para alcançar o minimo de propostas diarias
6      if((peso >= corte || faltando){ // caso o peso seja maior que o minimo exigido ou falte para atingir meta diario
7        reduzMinAP(BD,faltando);   //reduz a quantidade de regras que faltam para atingir a meta diaria
8        // carrega na tabela de filas ordenadas o id da regra e seu peso
9        cadastraRegraOrdenada(BD,((lista->no)->id),peso);
10     }
11     else{ // caso não tenha o peso para entrar na lista a ser apresentada ao usuario, entra na lista de espera
12       cadastraFilaEspera(BD,((lista->no)->id));
13     }
14    atualizaListaDeRegras(lista); //retira o nó ja ponderado e passa para proximo nó
15   }
16 }
```

Figura 7.12: Filtra Lista Fonte: Elaborado pelo autor

A função apresentada na Figura 7.12 consiste em um loop que é executado uma vez para cada membro na lista recebida. Para cada nó nessa lista é feito o seguinte procedimento: recupera o valor do peso da regra em questão (peso), o valor de corte para propor uma regra (corte) e quantas regras faltam para a meta diária de propostas (faltando).

Caso o peso da regra seja maior ou igual ao valor mínimo definido por corte ou caso ainda não tenha sido alcançada a meta diária de propostas, o sistema chama a função reduzMinAp() e a cadastraRegraOrdenada. A primeira reduz em um o valor da quantidade de regras que faltam para cumprir a meta diária. A segunda cadastra na tabela de regras ordenadas o id da possível regra assim como seu peso. Caso contrário, a regra é armazenada na tabela de fila de espera através da função cadastraFilaEspera().

A função ordenaRegras() da Figura 7.7 finaliza-se com um teste condicional que ocorre caso a cota mínima para ordenação não tenha sido alcançada. Este teste condicional, na linha 26 da Figura, testa se a meta de propostas diárias foi alcançada. Caso não tenha sido alcançada a função manipulaListaEspera() é chamada. Essa última função cria uma lista

com as funções na lista de espera, ponderando seus valores, em seguida chama a função filtraLista() passando a lista criada.

5. Proponente de Regras

Esse é o módulo responsável por apresentar ao usuário as propostas que foram criadas e pré-selecionadas nos módulos anteriores. Assim esse é o último módulo pelo qual uma regra passa antes de ser aprovada. Esse módulo é relativamente mais simples que os anteriores, uma vez que ele apenas envia as propostas pré-selecionadas ao usuário e caso esse aprove, cria a regra, caso contrário a regra é colocada na tabela de regras negadas. O fluxo desse módulo é visto na Figura 7.13.

```
apresentaRegrasPropostas(){
    while(true){
      acessaKnowledge(BD);
      nextId = consultaProximoIDPropostas(BD);
      lestId = consultaUltimoIdRegrasOrdenadas(BD);
      while(nextId < lestId ){
        // recupera dados da tabela regras ordenadas
        id = consultaRegraOrdenada(BD,"idpossivelregra",nextId);
        peso = consultaRegraOrdenada(BD,"peso",nextId);
        //recupera dados da possivel regra
        tipoSensor = consultaPossivelRegra(BD,"tipoSensor",id);
        idSensor = consultaPossivelRegra(BD,"idSensor",id);
        tipoComparacao = consultaPossivelRegra(BD,"tipoComparacao",id);
        valorComparado = consultaPossivelRegra(BD,"valorComparado",id);
        tipoEquipamento = consultaPossivelRegra(BD,"tipoEquipamento",id);
        idEquipamento = consultaPossivelRegra(BD,"idEquipamento",id);
        valorPassado = consultaPossivelRegra(BD,"valorPassado",id);
        periodo = consultaPossivelRegra(BD,"periodo",id);
        //consulta usuario solicitando aprovar a regra
        resposta = propoeRegra(tipoSensor,idSensor,tipoComparacao,valorComparado,tipoEquipamento,idEquipamento,valorPassado);
        //atualiza dados para qualidade total
        atualizaNegadasAceitas(BD,periodo,resposta);
        nextId = converteInt2(converteInt(nextId)+1);
        atualizaUltimoApresentado(BD,nextId);
        media = pesoMedio(BD,resposta);
        qntdMedia = qntdProposta(BD,resposta);
        media = (media*qntdMedia + peso)/(qntdMedia+1);
        qntdMedia = qntdMedia+1;
        atualizaMedia(BD,resposta,media);
        atualizaQntd(BD,resposta,qntdMedia);
      } fechaCon(BD); Sleep(5000);
    }
}
```

Figura 7.13: Proponente de Regras Fonte: Elaborado pelo autor

A função apresentada na Figura 7.13 é descrita em quatro partes: i) dentro de um *loop* infinito abre-se uma conexão com o banco; carrega uma variável com o id da última proposta apresentada ao usuário somando mais um, ou seja, a próxima proposta que deve ser apresentada (nextId); e uma com o id da última proposta salva na lista ordenada; ii) então um loop é executado até que todas as propostas da lista ordenada sejam enviadas ao usuário. Para cada uma dessas regras a função carrega as variáveis que compõem a regra; iii) é feito uma chamada da função propoeRegra() que envia a regra para aprovação do usuário e retorna uma resposta se a regra foi aprovada ou negada; e iv) por fim o sistema atualiza os valores de quantidade de regras negadas ou aceitas dependendo de qual foi a resposta do usuário e re-calcula a média do total para esse tipo de resposta.

Quando todas as regras disponíveis terminam de ser apresentadas, o sistema fecha a conexão com o banco e faz uma pausa antes de recomeçar o processo em i.

6. Qualidade Total

Como visto, a forma de avaliação de desempenho adotada é uma sequência de cálculos feitos pelo módulo qualidade total. Esses cálculos consideram a quantidade de sugestões apresentadas ao usuário, quais dessas propostas foram aceitas e quais foram rejeitadas; buscando uma otimização desse resultado. Esses cálculos representam a aprendizagem do agente, que refina suas decisões embasado nas suas experiências.

Esses testes são feitos para cada período separadamente e de forma sequencial. De forma que nunca dois testes de qualidade são efetuados ao mesmo tempo. Dessa forma, dentro de uma frequência de tempo pré-determinada o módulo qualidade total faz os testes de cada período definido, recalculando quando necessário as variáveis de controle do sistema.

A Figura 7.14 apresenta a lógica utilizada para verificar a necessidade de cálculos das variáveis de controle para cada um dos períodos. Também possui as chamadas de funções que fazem o cálculo do novo valor ideal dessas variáveis.

```
1  qualidadeSimples(periodo){
2  acessaKnowledge(BD);
3  apresentado = totalAguardando(BD);
4  apresentado2 = ultimoApresentado(BD);
5  cota = cotaLista(BD);
6  acc = consultaQualidadeAceito(BD,periodo);
7  dny = consultaQualidadeNegado(BD,periodo);
8  if( (apresentado2+cota) < apresentado ){
9   defineCorte();
10 }
11 if( (acc > 10) || (dny > 10) ){
12  ajustaFrequencia(BD,acc,dny);
13 }
14 //quantidade de propostas feitas por periodo
15 if( consultaFimPeriodo(BD,periodo)){
16  metaPeriodo(BD,periodo);
17 }
18 fechaCon(BD);
19 }
```

Figura 7.14: Qualidade Simples Fonte: Elaborado pelo autor

Essa função é dividida em quatro partes, i) abre o acesso ao banco de conhecimento e carrega as variaveis necessarias para os testes de qualidade; ii) testa se a quantidade de propostas apresentadas atingiram a cota de ordenação conforme discutido na Fórmula (3.10), caso o teste seja verdadeiro a nota mínima de corte é atualizada através da função defineCorte(); iii) testa se a quantidade de regras aceitas ou de regras negadas é maior que dez, caso uma das duas seja, chama a função para calcular uma nova quantidade de frequência mínima e a cota de ordenação; e iv) testa se o período terminou, caso tenha

terminado chama a função metaPeriodo() que verifica se a quantidade de regras propostas do período em questão está dentro de um intervalo mínimo e máximo aceitável.

A função descrita na parte ii, é representada pela lógica apresentada na Figura 7.15, que tem o intuito de replicar a Fórmula (3.18).

```
1  defineCorte(){
2   //media dos pesos aceitos
3   positivo = consultaMedia(BD,"aceito"));
4   //media dos pesos negados
5   negativo = consultaMedia(BD,"negado"));
6   //media de corte
7   corteA = consultaMedia(BD,"corte"));
8   // numero de regras aceitas
9   PosQ = consultaQntd(BD,"aceito");
10  // numero de regras negadas
11  NegQ = consultaQntd(BD,"negado"));
12  // numero de regras na media de corte
13  CorteQ = consultaQntd(BD,"corte"));
14  if(negativo < positivo){
15   newCorteQ = (CorteQ+NegQ); // define o numero de regras para calcular novo peso
16   corteA = ((corteA*CorteQ) + (negativo*negQ*0.9))/newCorteQ; // determina o novo peso de corte
17  }
18  else{
19   newCorteQ = (CorteQ+posQ);// define o numero de regras para calcular novo peso
20   corteA = ((corteA*CorteQ) + (positivo*posQ*0.7)) /newCorteQ; // determina o novo peso de corte
21  }
22  if(corteA >9.9){corteA = 9.5;}
23   atualizaUltimoApresentado(BD,apresentado);
24   atualizaCorte(BD,corteA);
25   atualizaQuantidadeCorte(BD,newCorteQ);
26 }
```

Figura 7.15: Define Corte Fonte: Elaborado pelo autor

A cada reordenação a nota de corte deve ser reavaliada, para possibilitar a evolução dos resultados do sistema. Assim, sempre que a condição da Fórmula (3.10) é satisfeita executa-se o defineCorte(). Devido aos testes serem feitos de forma sequencial dentre os períodos, o primeiro período que for avaliado quando essa condição for satisfeita, será o responsável por fazer essa atualização chamando a função apresentada na Figura 7.15. A estrutura sequencial garante o controle sobre esse cálculo, certificando que sua frequência seja exatamente a mesma que da ordenação de regras.

A função supracitada carrega seis variáveis que são utilizadas para o cálculo da função (3.18), que determina qual o novo valor da nota de corte. Essas seis variáveis são a média atual das propostas aceitas (positivo), a média atual de propostas negadas (negativo), a média atual de corte (corteA), a quantidade de propostas aceitas (PosQ), a quantidade de propostas negadas (NegQ), e a quantidade de propostas usadas para gerar a média de corte atual (CorteQ).

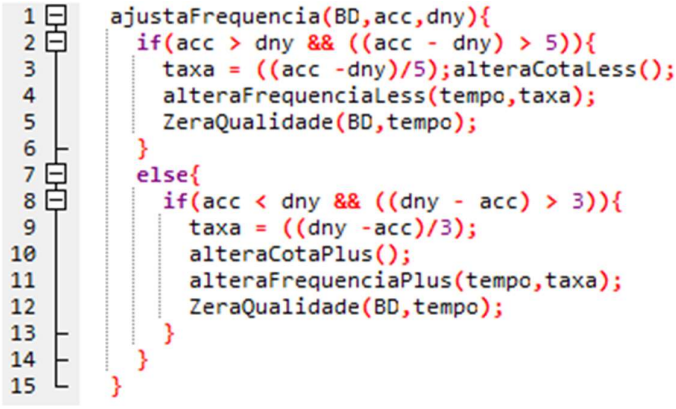
Em seguida, a função testa qual a maior média, dos negados ou dos aceitos (linha 14), caso a média de resposta negadas seja menor que das aceitas, aplica-se a Fórmula (3.18a) caso contrário aplica-se (3.18b).

Com a nova nota de corte, o sistema verifica se a média está maior que nove e nove décimos (o que significa próximo do máximo possível). Caso esteja, para evitar um travamento

de surgimento de novas propostas, o sistema atribui o valor de nove e meio a nota de corte, caso contrário a nota de corte permanece com o valor calculado.

Com o valor para a nota de corte definido, a função atualiza o valor do id referente ao último cálculo de qualidade (para o correto funcionamento da Fórmula (3.10)), atualiza o valor para nova nota de corte e a quantidade de regras usadas para calcular a média de corte.

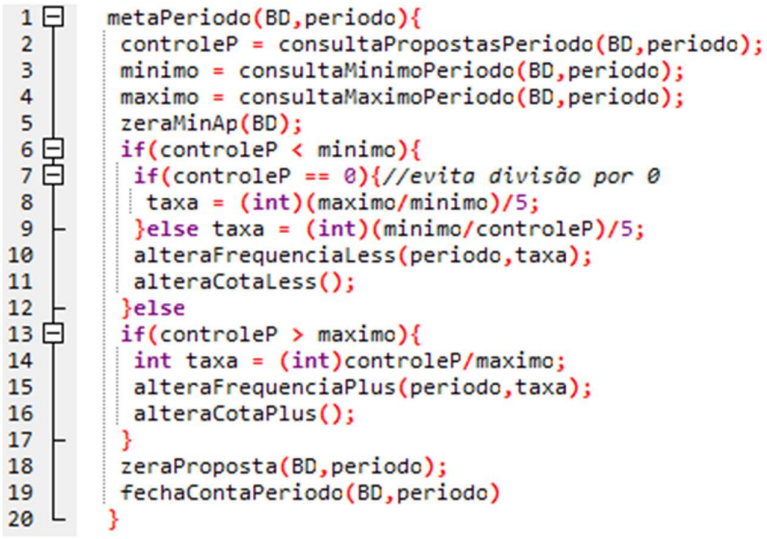
Outra função chamada pela função qualidadeSimples() é a ajustaFrequencia() que atua buscando o número ideal de repetições para criar-se uma proposta de regra. Sua lógica é vista na Figura 7.16.Os testes condicionais nas linhas 2 e 8 reproduzem o efeito apresentado de escolha da Fórmula (3.18) e o corpo dos testes condicionais reproduzem os cálculos.



```
1  ajustaFrequencia(BD,acc,dny){
2    if(acc > dny && ((acc - dny) > 5)){
3      taxa = ((acc -dny)/5);alteraCotaLess();
4      alteraFrequenciaLess(tempo,taxa);
5      ZeraQualidade(BD,tempo);
6    }
7    else{
8      if(acc < dny && ((dny - acc) > 3)){
9        taxa = ((dny -acc)/3);
10       alteraCotaPlus();
11       alteraFrequenciaPlus(tempo,taxa);
12       ZeraQualidade(BD,tempo);
13     }
14   }
15 }
```

Figura 7.16: Ajuste de Frequência Fonte: Elaborado pelo autor

Em seguida na qualidadeSimples() é feito o teste encerramento de período. Caso o resultado seja positivo chama a função metadePeriodo(). Seu objetivo é verificar se a quantidade de propostas criadas pelo agente simples desse período está dentro de um intervalo mínimo e máximo esperado. Essa função objetiva cumprir a implementação da Fórmula (3.20) e sua lógica é vista na Figura 7.17.



```
1  metaPeriodo(BD,periodo){
2   controleP = consultaPropostasPeriodo(BD,periodo);
3   minimo = consultaMinimoPeriodo(BD,periodo);
4   maximo = consultaMaximoPeriodo(BD,periodo);
5   zeraMinAp(BD);
6   if(controleP < minimo){
7    if(controleP == 0){//evita divisão por 0
8     taxa = (int)(maximo/minimo)/5;
9    }else taxa = (int)(minimo/controleP)/5;
10   alteraFrequenciaLess(periodo,taxa);
11   alteraCotaLess();
12  }else
13  if(controleP > maximo){
14   int taxa = (int)controleP/maximo;
15   alteraFrequenciaPlus(periodo,taxa);
16   alteraCotaPlus();
17  }
18  zeraProposta(BD,periodo);
19  fechaContaPeriodo(BD,periodo)
20 }
```

Figura 7.17: Meta Período Fonte: Elaborado pelo autor

A lógica da função metaPeriodo() é explicada separando-a em três partes: i) carrega as seguintes variáveis usadas para os cálculos: a quantidade de propostas feitas pelo agente simples daquele período (ControleP); a quantidade mínima de propostas esperadas para aquele período (mínimo); e a quantidade máxima de propostas esperadas para aquele período (máximo). Após carregar essas variáveis, a quantidade mínima de propostas a serem apresentadas é restaurado ao seu valor original no KDB; ii) é feito um teste condicional, se a quantidade de propostas do período for menor que o mínimo aplica-se a Fórmula (3.21), caso a quantidade de propostas do período seja superior ao máximo, aplica-se a Fórmula (3.22); iii) a quantidade de propostas feitas para aquele período salva no KDB recebe zero para iniciar uma nova contagem e a última ação antes da função se encerrar é alterar a flag de atualização de período, informando que as contas do período já foram feitas.