Sistemas pervasivos integrados por agentes inteligentes em JASON e Raspberry Pi

Aloisio Kneipp dos Santos[1], **Alexandre Zamberlan**[12]

[1]Universidade Franciscana (UFN)
Santa Maria – RS

[2]Laboratório de Práticas da Computação
Universidade Franciscana (UFN)
Santa Maria – RS

{aloisio.santos,alexz}@ufn.edu.br

Resumo. *Esta pesquisa integra a teoria BDI (Belief, Desire, Intention) de agentes inteligentes com os fundamentos da Computação Pervasiva, no contexto da "Internet das Coisas"(do inglês, Internet of Things - IoT), em que dispositivos possam comunicar-se entre si de forma inteligente, autônoma, proativa e flexível. Portanto, esta pesquisa projetou e desenvolveu uma simulação de solução computacional que fornece integração, no contexto IoT, entre dispositivos Raspberry Pi e a tecnologia de Sistemas Multiagentes implementados em JASON. Para isso, a pesquisa tem caráter exploratório com revisão bibliográfica apoiada em estudo de caso. Foram utilizados a metodologia PROMETHEUS para modelagem de Sistemas Multiagentes, as linguagens de programação Java e AgentSpeak(L), o interpretador JASON e o computador Raspberry Pi.*

Abstract. *This research integrates the BDI (Belief, Desire, Intention) theory (intelligent agents) with fundamentals of Pervasive Computing (Internet of Things - IoT), where devices can communicate intelligently, autonomously, proactively and flexibly. Therefore, we focus on the design and development of a computational solution that provides integration (simulation process), in the IoT context, between Raspberry Pi devices and the Multi-Agent Systems technology implemented in JASON. The research has an exploratory character with bibliographic review supported by a case study. We have used the PROMETHEUS methodology for modeling Multi-Agent Systems, the Java and AgentSpeak (L) programming languages, the JASON interpreter and the Raspberry Pi computer.*

1. Introdução

A computação é uma realidade em residências, empresas, indústrias, hospitais, escolas, por exemplo. Os diversos dispositivos eletrônicos existentes nesses cenários, certamente, estarão comunicando-se entre si, agilizando diferentes tarefas e compartilhando recursos diversos. E toda essa interação cria espaços computacionais inteligentes, onde os computadores e/ou dispositivos dotados de processamento e conexão à Internet podem oferecer serviços aos usuários, não importando o local e nem a hora (um dos princípios da área Internet das Coisas - IoT) [Zamberlan et al. 2014].

De acordo com alguns dos principais autores da área, [Weiser 1991] e [Saha and Mukherjee 2003], assume-se que a Computação Pervasiva, no contexto de IoT, é a computação disponível em qualquer lugar, a qualquer tempo, e que o usuário possa usar qualquer dispositivo para ter acesso ao seu ambiente computacional. Sabendo-se, também, que as características básicas de agentes inteligentes são autonomia, mobilidade e proatividade, acredita-se que isso aproxima essas áreas. Aproximar a teoria BDI (*Belief*, *Desire*, *Intention*) de agentes e Sistemas Multiagentes cognitivos no contexto da Computação Pervasiva pode ser concretizada pela integração da linguagem de especificação AgentSpeak(L) e seu ambiente de interpretação JASON [Zamberlan et al. 2014].

Integrar Computação Pervasiva com Sistemas Multiagentes Inteligentes (IoT) pode gerar resultados promissores. Isto é, enquanto a pervasividade agrega à habilidade de perceber o ambiente, agentes somam quanto à autonomia e à proatividade. Além disso, são capazes de gerar comunicação entre seus elementos. Por isso, acredita-se, com esta pesquisa, que a área de Sistemas Multiagentes pode ser um meio eficaz de operacionalizar a sistemas pervasivos. Também, que o uso da teoria BDI, por meio da linguagem AgentSpeak(L) e seu interpretador JASON, pode ser fundamental para a integração dessas áreas, uma vez que o JASON fornece recursos de comunicação e concorrência entre os agentes. A área de Sistemas Multiagentes encontra-se consolidada, com metodologias e ferramentas para projeto e implementação de sistemas inteligentes.

Dessa forma, este trabalho teve como objetivo principal projetar e desenvolver uma simulação para uma solução computacional que forneça integração, no contexto IoT, entre dispositivos Raspberry Pi e a tecnologia de Sistemas Multiagentes implementados em JASON. E como objetivos específicos, assumiram-se: entender e realizar testes de integração entre JASON e Raspberry Pi; entender o processo de controle de um Raspberry Pi via conexão Internet e a linguagem Java, que é a base do interpretador JASON; modelar os agentes que irão representar os dispositivos, tornando-os inteligentes no contexto IoT; definir a arquitetura distribuída de funcionamento do sistema pervasivo de integração entre agentes e dispositivos; transferir os comportamentos de agentes modelados para dispositivos Raspberry Pi.

Registra-se que a pesquisa pode gerar produto ou serviço inovadores, pois são baseados em uma arquitetura genérica e flexível que poderá ser adaptada em ambientes diversos. Como por exemplo, instituições de ensino, com controle de presença do aluno por geolocalização via celular; controle de iluminação e áudio da sala de aula; controle da climatização da sala; gestão da distribuição de documentos eletrônicos entre os presentes na sala, entre outros.

2. Revisão bibliográfica

Nesta seção, são apresentados conceitos e alguns trabalhos relacionados às teorias de Sistemas Pervasivos-Ubíquos e Sistemas Multiagentes. Também são discutidos a linguagem AgentSpeak(L), seu interpretador JASON e o Raspberry Pi.

2.1. Computação pervasiva e ubíqua - Internet das Coisas

Computação Pervasiva e Ubíqua, no contexto de IoT, é a computação disponível em qualquer lugar, a qualquer tempo, e que o usuário possa usar qualquer dispositivo para ter

acesso ao seu ambiente computacional [Weiser 1991] por meio do modelo TCP/IP de Redes de Computadores. Muitos consideram os termos sinônimos, entretanto, o que os diferencia é a mobilidade dos dispositivos. Dessa forma, quanto mais móvel, mais ubíquo, quanto mais fixo, mais pervasivo.

Computação pervasiva, segundo [Weiser 1991] nos anos 90, "é a computação que estará presente em etiquetas de roupas, xícaras de café, interruptores de luz e que será transparente (imperceptível) aos usuários". A computação ubíqua é a computação não fixa (ou móvel), ou seja disposta em todo e qualquer lugar que possa ser captado informações, utilizando sensores e/ou utilizando atuadores. Ao combinar a computação pervasiva com a computação ubíqua e com protocolos do modelo TCP/IP é possível operacionalizar IoT.

As aplicações pervasivas e ubíquoas podem ser proativas, isto é, podem identificar o que o usuário deseja e providenciar ações devidas no momento correto. Nesses processamentos, os sistemas utilizam-se do contexto formado pelas informações relacionadas ao ambiente, seja por meio de sensores, seja pela geração ou envio a partir de algum dispositivo. Para que aplicações pervasivas funcionem adequadamente, é essencial que a percepção de contexto e a execução de tarefas sejam características básicas [Helal et al. 2005]. Segundo [Pereira and Librelotto 2012], essa percepção do contexto e a capacidade de utilizá-lo para executar tarefas são o que caracterizam uma aplicação sensível ao contexto (*context-awareness*) e também um sistema multiagente, conforme [Bordini et al. 2007].

2.2. Sistemas multiagentes - SMA

Na área de Inteligência Artificial, há a subárea de Inteligência Artificial Distribuída, em que problemas são resolvidos de maneira descentralizada por alguns elementos responsáveis por tarefas específicas. Dessa forma, esses elementos são conhecidos como agentes e estão inseridos em sociedades (sistemas), interagindo e colaborando para resolver problemas em comum [Bordini et al. 2007]. Os agentes são modelados e implementados para terem comportamentos autônomos, proativos, adaptáveis e de interação social (comunicação entre agentes) [Hübner and Bordini 2010]. Por estarem inseridos numa sociedade, compartilham o ambiente, objetivos e recursos, por isso a sociedade multiagente é tida com comportamento emergente de inteligência [Zamberlan et al. 2014].

A teoria SMA está presente em muitas aplicações, como simulação computacional, jogos eletrônicos e sistemas autônomos robóticos, por exemplo. Essas aplicações têm em comum componentes ou elementos (software ou hardware) que precisam atuar e perceber o ambiente, sem que sejam controlados por algo ou alguém (autonomia), que executem ações ou tarefas de forma proativa, adaptáveis e flexíveis às situações inesperadas do ambiente. Além disso, precisam estar em constante comunicação entre si, coordenando tarefas ou ações junto ao ambiente.

2.2.1. AgentSpeak(L) e JASON

Há inúmeras linguagens, interpretadores, ambientes de desenvolvimento e metodologias para modelar e implementar Sistemas Multiagentes. Porém, a linguagem AgentSpeak(L) e seu interpretador JASON vêm tendo destaque nesse universo. A linguagem e seu interpretador, contudo, obedecem a princípios da teoria BDI (*Belief-Desire-Intention*) de

agentes inteligentes [Bordini et al. 2007].

Conforme apresentado em [Zamberlan et al. 2014], um agente na linguagem AgentSpeak(L) corresponde à especificação de um conjunto de crenças (sua base de conhecimento) e um conjunto de planos. Tanto uma crença quanto um plano são expressos como predicados[1]. Os planos fazem referência a ações básicas que um agente é capaz de executar em seu ambiente. Essas ações são definidas por atributos com síımbolos predicativos especiais. Um plano é formado por um evento ativador, seguido de uma condição ou contexto (crenças). O contexto deve disparar ou não a execução do plano. O restante do plano é um conjunto de ações básicas no ambiente, ou sub-objetivos, ou diretivas de comunicação entre agentes [Bordini et al. 2007].

Segundo [Helal et al. 2005], a operacionalização de um ambiente pervasivo depende basicamente de três requisitos fundamentais. Porém, neste trabalho, já se definiu como operacionalizar esses requisitos: um sistema de automação contendo sensores, atuadores, controladores e interfaces humano-computador, via Raspberry Pi; uma infraestrutura de comunicação de dados para troca de informações entre dispositivos e entre usuários desse ambiente, também via Raspberry Pi; sistemas inteligentes, cientes de contexto, adaptáveis, evolucionários e capazes de atender as necessidades dos usuários em suas atividades diárias, oferecendo suporte inteligente, por meio da ferramenta JASON.

2.3. Raspberry PI

É um micro computador de placa única (*SoC - System On Chip*) com tamanho reduzido (tamanho de um cartão de crédito), que possui no mínimo um processador 1.2 GHz, ARM Cortex-A53, 1 GB de de memória RAM, Bluetooth 4.1, placa de rede Fast Ethernet e WI-FI 2.4 GHz.

Raspberry Pi é considerado de baixo custo, possui tamanho reduzido, é executado sobre o sistema Raspibian, baseado no GNU/Linux Debian. Por possuir um processador incorporado, tem possibilidade de execução de instruções. Por possuir a interface GPIO, *General Purpose Input/Output*, pode ser adicionado diversos sensores e módulos para receber informações do ambiente. Não é restrito a utilização de fios, sendo possível utilizar módulos *Wi-Fi*.

2.4. Relés de Acionamento

De acordo com [Mattede 2020], os relés são dispositivos elétricos que tem como função produzir modificações repentinas, mas predeterminadas em um ou mais circuitos elétricos de saída. O relé tem um circuito de comando, que no momento em que é alimentado por uma corrente, aciona um eletroímã (bobina) que faz a mudança de posição de outro par de contatores, que estão ligados a um circuito ou comando secundário. Ou seja, um relé se configura como um contato que abre e fecha de acordo com umas configuração. Há vários tipos de construções mecânicas para relés. Neste trabalho, focou-se no relé eletromecânico que possui uma parte móvel que é acionada por um campo eletromagnético gerado por uma bobina.

O Sonoff Basic possui internamente um microcontrolador com *socket Wi-Fi* embutido, que tem a função de se conectar à rede sem fio. Também possui uma fonte bivolt

[1]Como acontece na linguagem Prolog.

que pode ser ligada diretamente na rede elétrica, seja ela 110 ou 220 Volts. Com isso, o Sonoff Basic torna-se um interruptor inteligente sem fio, versátil e de baixo custo.

Utiliza o protocolo *Message Queue Telemetry Transport* (MQTT[2]), criado em meados de 1999 pela IBM. Esse protocolo é de comunicação máquina-máquina, que possuía como o objetivo interconectar sistemas de telemetria de oleodutos via satélite, visando a segurança na transmissão do pacote de dados. Os clientes do protocolo MQTT podem ser *publisher* e/ou *subscribers*, sendo que quando atua de maneira *publisher* o cliente envia informações para o *broker*, e quando atua como *subscriber* se associa para receber as informações. O *broker* receber e armazena as informações do *publisher* e envia para os *subscribers*.

De acordo em [Vanzella 2018] o protocolo MQTT possui três níveis de qualidade de serviço *QoS*: nível 0: em que a mensagem será enviada apenas uma vez e que a mensagem pode não ser recebida; nível 1: onde o protocolo garante que a mensagem será entregue pelo menos uma vez, podendo ser retransmitida mais de uma vez, fazendo com que o destinatário receba a mensagem duplicada; nível 2: é o nível mais alto, onde é enviada a mensagem e há verificação se a mensagem está sendo recebida apenas uma vez. Destaca-se que o meio físico de comunicação é *Wi-Fi*.

2.5. Metodologia PROMETHEUS

Uma metodologia para o projeto e a implementação do SMA deve obedecer às orientações da Engenharia de Software Orientada a Agentes (*Agent-Oriented Software Engineering* - AOSE). Assume-se que uma metodologia de projeto e de construção de SMA deve conter um conjunto de métodos, processos e ferramentas para o desenvolvimento de um sistema baseado em agentes, sendo que a modelagem deve contemplar objetivos do sistema, papéis e interações dos seus elementos [Padgham and Winikoff 2004]. A metodologia PROMETHEUS possui as etapas: especificação do sistema: descrição dos objetivos e cenários do sistema. Para isso, elenca as principais funcionalidades; projeto arquitetural: definição da Visão Geral do sistema. Os agentes são organizados em papéis e são descritas suas relações; projeto detalhado: detalha-se, então, todos os agentes e seus papéis. Descrevem-se eventos, planos e dados.

2.6. Trabalhos relacionados

Nesta seção, buscou-se apresentar e discutir trabalhos que implementaram sistemas pervasivos ou ubíquos com características de sistemas inteligentes ou de ambientes inteligentes, seja com uso de SMA e Raspberry Pi, ou não.

O *framework*, proposto em [Perozzo 2011], define uma arquitetura que é implementada para atender as funcionalidades de mapeamento de dispositivos físicos de automação presentes em ambientes inteligentes[3] para o universo computacional; suporte ao desenvolvimento de aplicações interativas para o acesso aos serviços e aos dispositivos de automação dos ambientes inteligentes; criação de cenários de automação independentes de plataforma de hardware em que serão executados; reutilização de projetos para otimização do tempo de desenvolvimento de novas aplicações interativas; geração automática de código em que são constituídas aplicações baseadas no perfil do hardware e da linguagem suportada pelo *middleware* de interatividade.

[2]Em português, Transporte de Filas de Mensagem de Telemetria.

[3]Nesse trabalho, utilizou-se a sigla AmI para ambientes inteligentes.

O trabalho, apresentado em [Lazarin and Pantoja 2015], projetou e construiu uma plataforma de agentes robóticos com agentes de software embutidos em dispositivos de hardware como Raspberry Pi e Arduino. A plataforma tem inserida o *framework* JASON no Raspberry Pi, possibilitando o controle direto dos pinos referentes a sensores e atuadores do Arduino. Para isso, toda a comunicação entre os dispositivos foi implementada via a biblioteca Javino (comunicação entre Java e Arduino por porta serial). O trabalho também apresentou uma sugestão metodológica para desenvolvimento de agentes robóticos por meio de um estudo de caso. O *chassi* da imagem é uma estrutura pronta e adquirida comercialmente (que pode ser customizada). Essa estrutura contém o Rasberry Pi (o 'cérebro' ou hardware de controle), o JASON (a 'mente' ou software de controle) e o Arduino (gestão de atuadores e sensores ou 'olhos', 'mãos'). Na arquitetura proposta, o Raspberry Pi é o responsável pela simulação do agente e processamento de todas as informações. O Arduino, via Javino, realiza a comunicação entre a aplicação JASON do agente com o Arduino de maneira serial. Além disso, executa as funções de perceber (sensorar) as informações do ambiente e executar (atuar) ações no ambiente por meio de atuadores.

No trabalho prototipado por [Nunes et al. 2018], foi proposto a gestão inteligente de um laboratório (maquete) via Sistemas Autônomos (LISA), com uso de diferentes SMA e o *middleware* ContextNet para IoT embarcados. A proposta buscou administrar um laboratório controlando dispositivos eletrônicos tais como a porta, luz, arcondicionado. Foi obtido resultados positivos, em que o LED representativo do atuador do ar condicionado da sala acendia indicando seu acionamento quando o agente detectava uma temperatura pré-definida como alta. Neste trabalho foi utilizado agentes SMA, junto com o microcontrolador ATMEGA que realizaria o acionamento dos sensores e atuadores, com isso os dois agentes SMA estavam recebendo percepções um do outro, onde o agente executaria os planos pré programados em conformidade com as possíveis mensagens recebidas.

No trabalho realizado por [de Souza Junior 2018], foi projetado um sistema IoT para gerenciamento e controle de aquário marinho, a partir da leitura de sensores instalados no aquário e no ambiente, fornecendo informações dos parâmetros físicos e químicos do ecossistema ao aquarista. Foram utilizados no desenvolvimento do projeto sensores de temperatura e umidade relativa do ar DHT22, temperatura de líquidos DS1820, luz TSL2561, nível da água 9S44, e os seguintes acionadores: módulo relé 5v oito canais 4MD06, *display* LCD 16x2, LED 5mm, bomba de reposição de água, *chiller*, aquecedor e uma luminária. Utilizando o GPIO da Raspberry Pi foi possível efetuar a leitura das informações no ambiente, bem como acionar os relés responsáveis pelos atuadores nos momentos pré definidos. Com isso, foi possível obter uma baixa variação de temperatura da água, onde auxiliou diretamente na saúde dos animais, pois uma alta variação de temperatura ocasiona a aceleração ou redução do metabolismo dos peixes e acompanhar de maneira mais interativa via uma pagina Web, onde era exibido as informações recebidas do ambiente.

Esse trabalho, [Zamberlan et al. 2014], realizou uma revisão bibliográfica sobre Sistemas Pervasivos e Sistemas Multiagentes BDI, com intuito de implementar a interação dessas duas áreas. Os agentes/dispositivos de controle do ambiente foram especificados e implementados em AgentSpeak(L) via seu interpretador Jason. No trabalho, foi criada

uma arquitetura de integração entre a teoria BDI e a de Computação Pervasiva, e para isso foi prototipado, somente em software, uma dinâmica de automação de uma sala de aula inteligente com projetor multimídia, persianas automatizadas, sistema de iluminação e sistema de áudio, de forma que esses dispositivos fossem integrados gerando um ambiente inteligente.

Enfim, dos trabalhos apresentados e discutidos, registra-se que [Zamberlan et al. 2014] e [Lazarin and Pantoja 2015] têm maior afinidade com esta proposta. Primeiro, porque ambos utilizaram a linguagem AgentSpeak(L) e seu interpretador JASON. Segundo, os dois projetaram uma arquitetura de referência para reprodução de trabalhos, facilitando a implementação desta proposta. Já em [Lazarin and Pantoja 2015], destaca-se o uso de Rasberry Pi com JASON, foco desta pesquisa. Entretanto, neste trabalho pretende-se utilizar a biblioteca nativa *GPIO* do Rasberry Pi, ao invés de Javino e PI4J. Também, destaca-se o trabalho de [Perozzo 2011], pois a dinâmica da proposta e da modelagem do ambiente construído são a referência para este trabalho. Há inúmeros trabalhos em que foram implementados agentes em sistemas ubíquos ou pervasivos, de maneira virtualizada ou em conjunto com microcontroladores. Porém, não foram encontrados trabalhos que possuíam uma rede de Raspberry Pi atuando como agentes inteligentes, interagindo de maneira ubiqua ou pervasiva sobre um ambiente.

3. Proposta do Sistema Pervasivo Multiagente

Este trabalho é uma pesquisa de revisão bibliográfica apoiada em estudo de caso. Considerando a lacuna encontrada e a possibilidade do uso de SMA, Raspberry Pi, JASON em sistemas pervasivos, este trabalho busca a integração dessas tecnologias, gerando ambientes e seus dispositivos mais inteligentes.

3.1. Materiais e métodos

Em relação à realização da pesquisa, a metodologia *Scrum* [Silveira et al. 2012] foi utilizada, bem como a técnica *Kanban* para a gestão das atividades assumidas no cronograma. Os *sprints* foram semanais tendo como referência as funcionalidades mapeadas e inseridas na ferramenta Trello para kanban. Para o projeto do SMA foi utilizada a metodologia PROMETHEUS. Ferramentas ou materiais utilizados foram: Astah: ambiente para diagramação de aspectos funcionais e estruturais da solução; GitHub: repositório para os códigos construídos; AgentSpeak(L): linguagem para programação de agentes na teoria BDI(Belief, Desire, Intention) [Bordini et al. 2007]; JASON: interpretador para linguagem AgentSpeak(L) [Bordini et al. 2007]; Eclipse: ambiente de desenvolvimento (IDE - *Integrated Development Environment*); Raspberry Pi; micro computador; Prometheus Design Tools (PDT): ferramenta para a diagramação dos artefatos na metodologia PROMETHEUS.

Destaca-se para a construção de ambientes pervasivos, são necessários controles, sensores e atuadores, que nesta investigação estarão acoplados ou interligados ao computador Raspberry Pi. A decisão por usar Raspberry Pi é devida a sua capacidade de armazenamento e de processamento.

Finalmente, para a avaliação da proposta, uma simulação foi projetada, implementada e discutida.

3.1.1. Estudo de caso

Dentro do contexto do estudo de caso, há diagramas que ilustram o processo de interação do sistema pervasivo e a arquitetura de componentes. Registra-se que os diagramas foram customizados para melhor ilustrar o sistema pervasivo no contexto de SMA. A Figura 1 mostra algumas funcionalidades do sistema pervasivo e seus atores (representados também pelos seus respectivos agentes). Aqui, buscou-se destacar a relação de atores com funcionalidades e seus respectivos agentes no ambiente SMA.

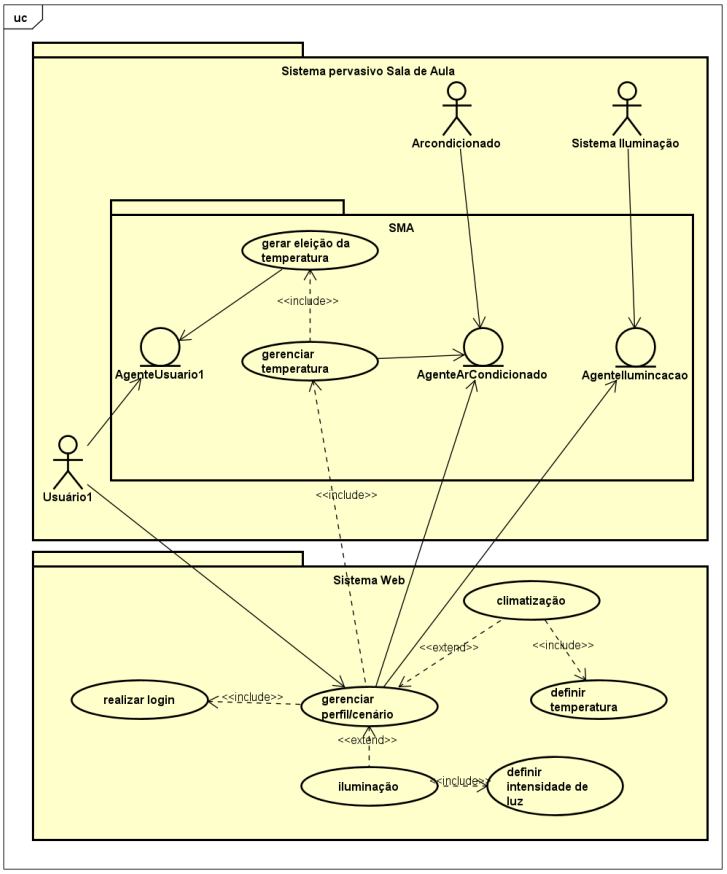

Figura 1. Diagrama de Casos de Uso.

Já na Figura 2, é representada a estrutura de componentes do sistema pervasivo. O destaque se dá para a 'interface' agente que representa tanto o usuário, quanto os dispositivos do ambiente, como sistemas de iluminação e climatização.

Referente à organização das tecnologias, a Figura 3 mostra como JASON, Raspberry Pi e relés de acionamento fazem a integração.

Em relação à metodologia PROMETHEUS, via PDT, apresenta-se o diagrama de Visão Geral do SMA (Figura 4), em que há todos os eventos ativadores (externos e internos) que os agentes precisam responder, seus respectivos planos para cada evento ativador e suas crenças iniciais.

A concepção inicial para o ambiente e as percepções em relação aos usuários será via rede *Wi-Fi* local, configurada no Raspberry Pi e os dispositivos móveis de cada

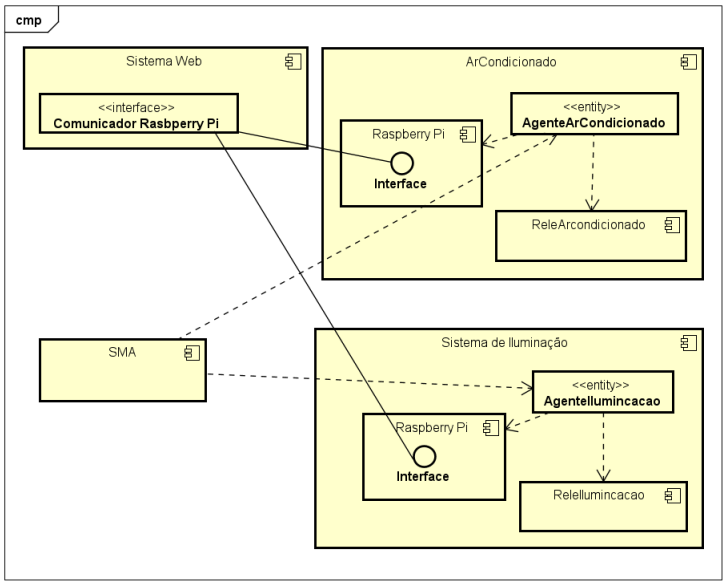

Figura 2. Diagrama de Componentes da arquitetura do Estudo de Caso.

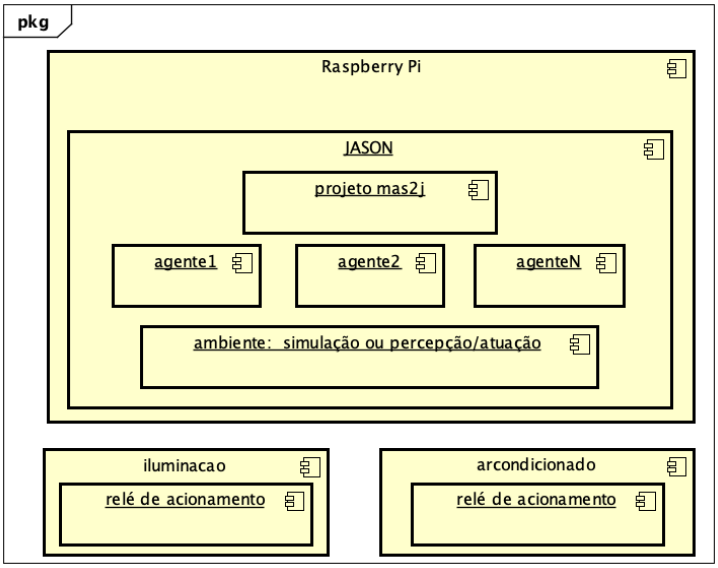

Figura 3. Diagrama de Componentes para integração entre tecnologias.

usuário. Dessa forma, quando um dispositivo conectar na rede local, automaticamente estará no ambiente do SMA proposto.

Registra-se, também, que a representação principal é um Raspberry Pi por ambiente que virtualiza e processa os comportamentos dos agentes, bem como o controle do ambiente (eventos ativadores). Apesar de ser um Raspberry Pi por ambiente, o sistema é descentralizado, pois os agentes são entidades autônomas, proativas, flexíveis e que tem comunicação.

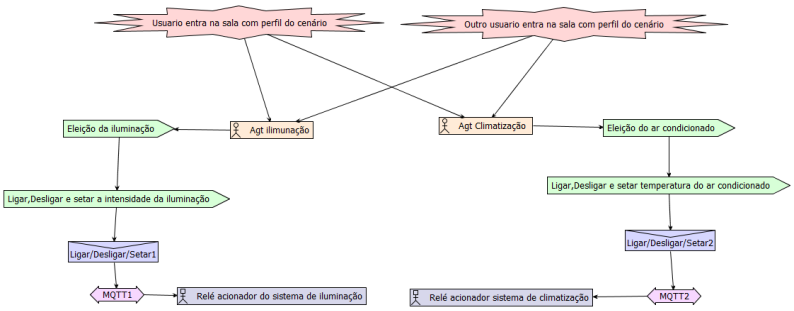

Figura 4. Diagrama de Visão Geral do SMA.

3.1.2. Sistema Web de gestão de perfis

O sistema Web tem a responsabilidade de gerenciar usuários, ambientes e perfis específicos de usuários em ambientes. Nesse sistema, é possível associar usuário a uma residência ou a um ambiente comercial e subdividi-lo em salas ou unidades. Cada sala, o usuário pode especificar um perfil de climatização e de iluminação, tendo como parâmetros intensidade da luz, temperatura, entre outros.

Os campos básicos do usuário são e-mail e senha de acesso, nome e tipo de usuário (administrador, comum, gestor). Os ambientes possuem como atributos descrição, localização (casa, apartamento, escritório, loja), iluminação (estágios - fraco, médio, forte), modo (inverno, verão, ventilar, secar) e temperatura. A relação usuário com climatização, por exemplo, possui os campos usuário, ambiente, horário (manhã, tarde, noite), modo e temperatura. Na Figura 5 ilustra a abstração. Assim, o sistema Web gera *webservice* do banco de dados que o Raspberry Pi pode consumir (via acesso remoto). Dessa forma, o sistema multiagente de automação pode controlar usuários e perfis. A ideia que o acesso ao *webservice* seja temporizado no *crontab*[4] do sistema operacional.

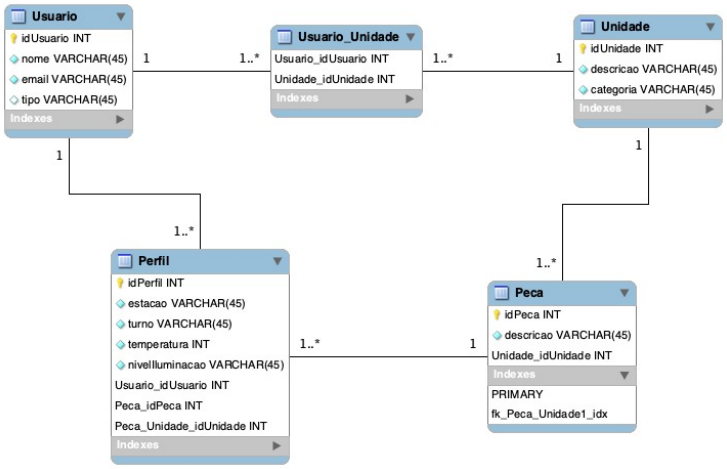

Figura 5. Diagrama Entidade-Relacionamento para o banco de dados do sistema Web.

[4]Serviço de agendamento de tarefas em Sistemas Operacionais Unix e Linux.

3.1.3. Sistema Multiagente em JASON

O SMA proposto deve contemplar situações como:

- um usuário sozinho na unidade ou sala: sistema deve configurar ou adaptar climatização e iluminação para o perfil desse usuário;
- dois usuários entrando na sala em um mesmo instante: sistema faz a média da temperatura e configura ou adapta sistema de climatização. Já em relação à iluminação, se ambos usuários possuírem o mesmo perfil, aplica-se o perfil, caso tenham perfis diferentes, o ambiente ficará na iluminação padrão (intensidade da luz média);
- *n* usuários em um instante: i) faz uma eleição e seleciona a temperatura mais preferida; ii) se os *n* usuários possuírem perfis de temperaturas diferentes entre si, o sistema faz a média;

A dinâmica representa uma situação em que usuários entram e saem de um ambiente. Porém, o sistema pervasivo precisa trabalhar com um espaço de tempo (aqui, define-se como instante), em que se o mesmo usuário entrar ou sair do ambiente, não haja alteração da configuração dos sistemas de climatização e de iluminação. Nesse período (espaço de tempo ou instante sem alteração), o ambiente pode até perceber/sensorar as entradas e as saídas do usuário, mas não dispara nenhuma ação de configuração, seja por eleição, seja por usuário único.

Decidiu-se que o instante na simulação são 2 segundos, enquanto que o espaço de tempo no sistema real seja 20 minutos. Lembrando que cada usuário tem seu perfil definido no sistema Web e carrega dados da relação usuário_climatização (qual a temperatura, em qual modo, em qual horário e em qual ambiente) e da relação usuário_iluminação (qual nível de iluminação, em qual horário e em qual ambiente).

3.1.4. Ambiente de testes e avaliações

O processo de avaliação do estudo de caso, que em termos gerais, foi por simulação. Simulou-se o gerenciamento de perfis de cenários (iluminação e climatização) e o gerenciamento de dispositivos móveis (que entram e saem da rede *Wi-Fi* do ambiente). Toda a simulação foi via interpretador JASON, que possui um console que verbaliza os eventos ativadores, execuções de planos e o inspecionador de crenças dos agentes. E a simulação executou no Raspberry Pi, comunicando com os relés de acionamento Sonoff. Dessa forma, foi possível acompanhar as interações e as decisões dos agentes, bem como visualizar essa interação no ambiente.

No início da simulação, dispara-se o método *init()* (Figura 6). Nesse método são sorteados os usuários que entram no ambiente, carregando suas preferências ou perfis para climatização e iluminação.

Durante a simulação, sempre que o método *executeAction()* for acionado, o sistema sorteia a entrada de novos usuários e a possibilidade da saída de usuário que já esteja no ambiente. Importante lembrar que o espaço de tempo mínimo seja mantido (instante). Na parte simulada do ambiente (arquivo *.java*), um usuário é um objeto com atributos de perfil para:

- climatização (nome do usuário, ambiente de perfil (sala, quarto, etc.), horário (manhã, tarde, noite), modo (verão ou inverno) e temperatura (faixa entre 18 graus e 35 graus);
- iluminação (nome do usuário, ambiente (sala, quarto, etc), nível de iluminação, horário (manhã, tarde, noite)).

```
105     /** Called before the MAS execution with the args informed in .mas2j */
106     @Override
107     public void init(String[] args) {
108         super.init(args);
109         try {
110             addPercept(ASSyntax.parseLiteral(sorteiaEstacaoTurno()));
111             //addPercept(ASSyntax.parseLiteral("contexto(verao,tarde)"));
112             addPercept(ASSyntax.parseLiteral(sorteiaUsuarioPeca()));
113         } catch (ParseException e) {
114             e.printStackTrace();
115         }
116     }
```

Figura 6. Código Java do ambiente de simulação que dispara crenças iniciais a todos agentes do ambiente.

Entretanto, na parte dos agentes (arquivos *.asl*), ou seja, na programação do comportamento dos agentes iluminação e climatização, o usuário também é um agente com crenças (Figura 7) e desejos/planos (Figura 8) sobre climatização e iluminação. Dessa forma, os agentes climatização e iluminação podem acessar essas crenças desses agentes usuários e iniciar o processo de configuração ou adaptação.

```
1  // Agent aloisio in project ambientePervasivo3
2
3  /* Os ambientes possuem como atributos descrição, localização (casa, apartamento, escritório, loja),
4   * iluminação (estágios - fraco, medio, forte), modo  (inverno, verao, ventilar, secar) e temperatura.
5   * Finalmente, a relação usuário com climatização, por exemplo,
6   * possui os campos usuário, ambiente, horario (manhã, tarde, noite), modo e temperatura.
7   */
8
9  /* Initial beliefs and rules */
10 //crencas vem do servidor e convertidas em arquivo texto
11
12 //CLIMATIZACAO
13 //peca1 - quarto de dormir
14 casa1_peca1_climatizacao(verao, noite, 24).
15 casa1_peca1_climatizacao(verao, tarde, 18).
16 casa1_peca1_climatizacao(inverno, noite, 26).
17 casa1_peca1_climatizacao(inverno, tarde, 21).
18
19 //peca2 - escritorio
20 casa1_peca2_climatizacao(verao, tarde, 22).
21 casa1_peca2_climatizacao(inverno, tarde, 22).
22
23 //ILUMINACAO
24 //peca1 - quarto de dormir
25 casa1_peca1_iluminacao(fraca,noite).
26 casa1_peca1_iluminacao(media,tarde).
27
28 //peca2 - escritorio
29 casa1_peca2_iluminacao(forte,manha).
30 casa1_peca2_iluminacao(forte,tarde).
31 casa1_peca2_iluminacao(forte,noite).
```

Figura 7. Base de crenças do agente *aloisio*.

Na modelagem de comportamento do agente iluminação, toda vez que ele perceber (entrada ou saída) um agente usuário, ele dispara planos para cada percepção realizada[5]:

- plano para a entrada de único agente usuário naquele instante;
- plano para entrada de mais de um agente usuário naquele instante;
- plano para saída de todos os usuários naquele instante;
- plano para saída de um usuário com outros usuários no ambiente: no intervalo sem alteração, o agente iluminação, apesar da saída de um usuário, não altera a sua configuração.

[5]Lembrando que um plano só dispara se o instante terminar, sendo que na simulação o instante dura 2 segundos.

```
/*initials plans */
+peca1(Agente): Agente == aloisio & contexto(Estacao,Turno)
    <-
        .print("vou avisar a peca1 o meu perfil");
         ?casa1_peca1_climatizacao(Estacao, Turno, Temperatura);
        .send(peca1_climatizacao,tell, casa1_peca1_climatizacao(Estacao,Turno,Temperatura));
        ?casa1_peca1_iluminacao(Estagio,Turno);
        .send(peca1_iluminacao,tell, casa1_peca1_iluminacao(Estagio,Turno)).

+peca1(Agente): Agente == aloisio & contexto(Estacao,Turno)
    <-
        .print("Nao tenho perfil para esta estacao ou turno").

+peca2(Agente): Agente == aloisio & contexto(Estacao,Turno)
    <-
        .print("vou avisar a peca2 o meu perfil");
         ?casa1_peca2_climatizacao(Estacao, Turno, Temperatura);
        .send(peca2_climatizacao,tell, casa1_peca2_climatizacao(Estacao,Turno,Temperatura));
        ?casa1_peca2_iluminacao(Estagio,Turno);
        .send(peca2_iluminacao,tell, casa1_peca2_iluminacao(Estagio,Turno)).

+peca2(Agente): Agente == aloisio & contexto(Estacao,Turno)
    <-
        .print("Nao tenho perfil para esta estacao ou turno").
```

Figura 8. Planos ou eventos ativadores do agente *aloisio*.

- plano para entrada de um ou mais usuários com usuários no ambiente: no intervalo sem alteração, o agente, apesar da entrada e saída de usuários, não altera a sua configuração.

Nas Figuras 9 e 10 são mostrados os planos que tratam dos eventos ativadores ou situações descritas. Na Figura 11, ilustra-se o comportamento do agente de climatização para gerenciar a entrada e saída de agentes usuários em determinado ambiente.

```
// Agent iluminacao in project ambientePervasivo3
/* Initial beliefs and rules */
/* Initial goals */
//!start.
/* Plans */
+!start : true <- .print("Sou um agente de iluminacao").

+peca1(Usuario): peca1(OutroUsuario) & Usuario \== OutroUsuario & casa1_peca1_iluminacao(OutroUsuario,EstagioOutroUsuario,Turno)
    <- .print("Notei que um segundo usuario ", Usuario, " entrou no recinto..... vou precisar negociar a iluminação");
       .wait(3000);
       ?casa1_peca1_iluminacao(Usuario,EstagioUsuario,Turno);
       !negociarIluminacao2(IluminacaoAtual,EstagioUsuario,EstagioOutroUsuario);
       .print("A iluminação negociada foi ....", IluminacaoAtual);
       .wait(1000);
       configurarIluminacao(IluminacaoAtual).

+!negociarIluminacao2(IluminacaoAtual,EstagioUsuario,EstagioOutroUsuario): EstagioUsuario == EstagioOutroUsuario
    <- IluminacaoAtual = EstagioUsuario.

+!negociarIluminacao2(IluminacaoAtual,EstagioUsuario,EstagioOutroUsuario): true
    <- IluminacaoAtual = media.

+peca1(Usuario): true
    <- .print("Notei que ", Usuario, " entrou no recinto.");
       .wait(300);
       ?casa1_peca1_iluminacao(Usuario,Estagio,Turno);
       .wait(1000);
       configurarIluminacao(Estagio).
```

Figura 9. Agente iluminação parte 1.

```
+peca1(Usuario): peca1(Usuario2) & Usuario \== Usuario2 & peca1(Usuario3) & Usuario \== Usuario3 & Usuario2 \== Usuario3 &
                 casa1_peca1_iluminacao(Usuario2,EstagioUsuario2,Turno) & casa1_peca1_iluminacao(Usuario3,EstagioUsuario3,Turno)
    <- .print("Notei que um terceiro usuario ", Usuario, " entrou no recinto..... vou precisar negociar a iluminação");
       .wait(3000);
       ?casa1_peca1_iluminacao(Usuario,EstagioUsuario,Turno);
       !negociarIluminacao3(IluminacaoAtual,EstagioUsuario,EstagioUsuario2,EstagioUsuario3);
       .print("A iluminação negociada foi ....", IluminacaoAtual);
       .wait(1000);
       configurarIluminacao(IluminacaoAtual).

+!negociarIluminacao3(IluminacaoAtual,EstagioUsuario,EstagioUsuario2,EstagioUsuario3): EstagioUsuario == EstagioUsuario2 &
                                                                                       EstagioUsuario2 == EstagioUsuario3
    <- IluminacaoAtual = EstagioUsuario.

+!negociarIluminacao3(IluminacaoAtual,EstagioUsuario,EstagioUsuario2,EstagioUsuario3): true
    <- IluminacaoAtual = media.

-peca1(Usuario): true
    <- .print("o ", Usuario, " saiu daqui.... vou decidir se irei ou não desligar a iluminação").
```

Figura 10. Agente iluminação parte 2.

Já em termos de resultados e análise desses resultados, nas Figuras 12, 13 e 14 são verbalizados os comportamentos da simulação, para: i) a entrada e saída de um usuário no ambiente; entrada de dois usuários e saída de um usuário; iii) entrada de três usuários

```
// Agent climatizacao in project ambientePervasivo3
/* Initial beliefs and rules */
/* Initial goals */
//!start.
/* Plans */
+!start : true <- .print("Sou um agente de climatizacao").

+peca1(Usuario): peca1(Usuario2) & Usuario \== Usuario2 & peca1(Usuario3) & Usuario \== Usuario3 & Usuario2 \== Usuario3 &
                 casa1_peca1_climatizacao(Usuario2,_,_,_,TemperaturaUsuario2) & casa1_peca1_climatizacao(Usuario3,_,_,_,TemperaturaUsuario3)
   <- .print("Notei que um terceiro usuario ", Usuario, " entrou no recinto..... vou precisar negociar a temperatura");
      .wait(3000);
      ?casa1_peca1_climatizacao(Usuario,Estacao,Turno,TemperaturaUsuario);
      TemperaturaAtual = (TemperaturaUsuario + TemperaturaUsuario2 + TemperaturaUsuario3)/3;
      .print("A temperatura negociada foi ....", TemperaturaAtual);
      configurarTemperatura(TemperaturaAtual).

+peca1(Usuario): peca1(OutroUsuario) & Usuario \== OutroUsuario & casa1_peca1_climatizacao(OutroUsuario,_,_,_,TemperaturaOutroUsuario)
   <- .print("Notei que um segundo usuario ", Usuario, " entrou no recinto..... vou precisar negociar a temperatura");
      .wait(3000);
      ?casa1_peca1_climatizacao(Usuario,Estacao,Turno,TemperaturaUsuario);
      TemperaturaAtual = (TemperaturaUsuario + TemperaturaOutroUsuario)/2;
      .print("A temperatura negociada foi ....", TemperaturaAtual);
      configurarTemperatura(TemperaturaAtual).

+peca1(Usuario): true
   <- .print("Notei que ", Usuario, " entrou no recinto.");
      .wait(300);
      ?casa1_peca1_climatizacao(Usuario,Estacao,Turno,Temperatura);
      configurarTemperatura(Temperatura).

-peca1(Usuario): true
   <- .print("o ", Usuario, " saiu daqui.... vou decidir se irei ou não desligar o arcondicionado").

```

Figura 11. Figura agente climatização.

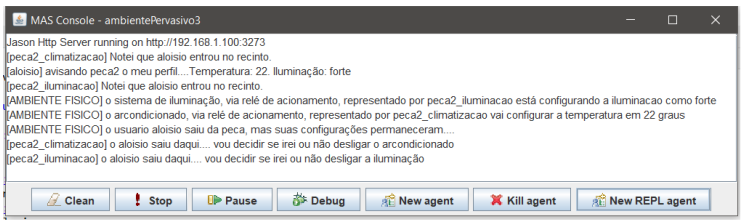

Figura 12. Resultado da simulação: entra um usuário e depois de um tempo, usuário sai.

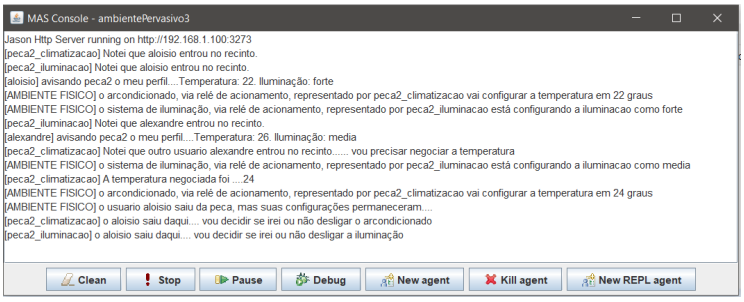

Figura 13. Resultado da simulação: entram dois usuários e sai um usuário depois de um tempo.

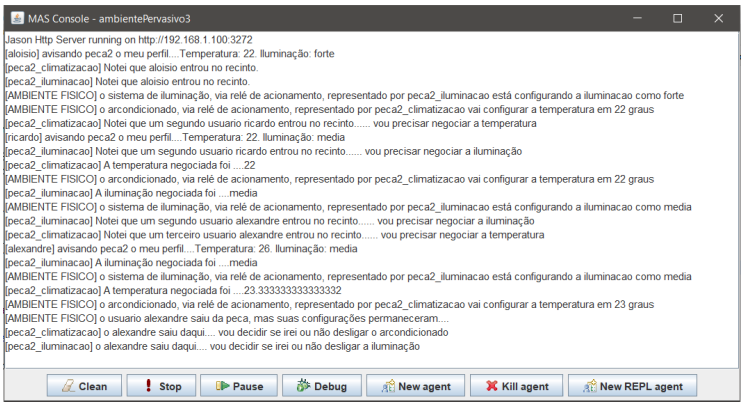

Figura 14. Resultado da simulação: entram três usuários e depois de um tempo, um usuário sai.

e saída de um usuário. Nas figuras, há indicação de cada agente e do ambiente físico que simula as entradas e saídas dos agentes.

Na Figura 15, é possível observar um cenário criado para atender às ações disparadas (arquivo *.java*) no planejamento do agente iluminação. Nesse cenário há 3 lâmpadas, conectas a relés de acionamento com endereço de rede local, para a comunicação com o processo de simulação. Quando a lâmpada halógena vintage acende, indica que é iluminação fraca. Quando duas lâmpadas normais acendem, mostra iluminação forte. E quando uma lâmpada normal acende mostra iluminação média.

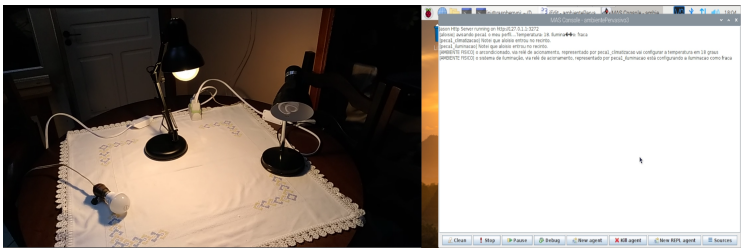

Figura 15. Cenário de simulação via lâmpadas. Neste exemplo, a lâmpada vintage está ligada, pois a simulação resulta em iluminação fraca.

4. Conclusões

Buscou-se apresentar e discutir assuntos referentes a sistemas pervasivos, ubíquos e multiagentes, sempre com o intuito de contextualizar e mostrar similaridades e diferenças entre eles. Dessa forma, a proposta obedece o modelo de um sistema pervasivo e multiagente, orientado a contexto (perfis ou cenários). Também foram apresentados e discutidos trabalhos relacionados, que forneceram uma base fundamental para a construção deste trabalho, uma vez que apresentam sugestões de tecnologias e arquiteturas. Também foram apresentados definições dos conceitos do mini computador Raspberry Pi e de Relés de acionamento. Apresentou-se os artefatos da modelagem para o estudo de caso, para que o leitor pudesse compreender as principais funcionalidades e questões estruturais do sistema pervasivo e multiagente em estudo.

Nesta proposta, sugere-se o uso de um Raspberry Pi por ambiente, pela otimização de custo. Ele tem a função de executar o sistema multiagente e fazer a comunicação entre agentes usuários, iluminação e climatização. Inclusive o dispositivo já foi utilizado para executar a simulação, comunicando-se com os relés de acionamento.

Registra-se que ainda não se sabe a forma de como perceber um agente no ambiente. Assim, sugere-se como trabalho futuro pesquisar e implementar o processo do ambiente detectar um usuário. Há algumas possibilidades: i) via *smartphone* ou *smartwatch* por conexão bluetooth, em que o Raspberry Pi tem ciclos de verificação de dispositivos *bluetooth*; ii) via *Wi-Fi* com implementação Multicast ou JGroups. Implementação de grupos e de rede virtual, em que o *smartphone* ou *smartwatch* acessam o ambiente e ao conectar na rede *Wi-Fi* do Raspberry Pi, automaticamente entram no grupo ou rede virtual Multicast, indicando sua presença.

Referências

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason*. Wiley.

de Souza Junior, R. F. (2018). *Sistema para Controle e Gerenciamento de Aquário Marinho Utilizando Automação por Raspberry Pi*. Centro Universitário Campo Limpo Paulista - Bacharelado em Ciência da Computação, Campo Limpo, SP.

Helal, S., Mann, W., El-Zabadani, H., King, J., Kaddoura, Y., and Jansen, E. (2005). The Gator Tech Smart House: A programmable pervasive space. *Computer*, 38(3):50–60.

Hübner, J. F. and Bordini, R. H. (2010). Using agent- and organisation-oriented programming to develop a team of agents for a competitive game. *Ann. Math. Artif. Intell.*, 59(3-4):351–372.

Lazarin, N. M. and Pantoja, C. E. (2015). A robotic-agent platform for embedding software agents using raspberry pi and arduino boards. *9th Software Agents, Environments and Applications School*, pages 13–20.

Mattede, H. (2020). O que é relé? como funciona um relé? https://www.mundodaeletrica.com.br/o-que-e-rele-como-funciona-um-rele/.

Nunes, P., de Almeida, I., Picanco, T., Pantoja, C., Samyn, L., de Jesus, V., and Manoel, F. (2018). Explorando a comunicaç ao entre sistemas multi-agentes embarcados em ambientes inteligentes para iot: Uma proposta de laboratório. In *XII Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações*, pages 238–243, Rio Grande, Brasil. SBC.

Padgham, L. and Winikoff, M. (2004). *Developing Intelligent Agent Systems: A practical guide.* John Wiley & Sons.

Pereira, H. and Librelotto, G. (2012). *ARCP: uma arquitetura para a utilização de computação nas nuvens nos ambientes de computação pervasiva.* Amazon.

Perozzo, R. F. (2011). *Framework para integração entre ambientes inteligentes e o sistema brasileiro de TV Digital*. PhD thesis, Universidade Federal do Rio Grande do Sul - UFRGS, Porto Alegre.

Saha, D. and Mukherjee, A. (2003). Pervasive computing: a paradigm for the 21st century. *Computer*, 36(3):25–31.

Silveira, P., Silveira, G., Lopes, S., Moreira, G., Steppat, N., and Kung, F. (2012). *Introdução à arquitetura e Design de Software*. Elsevier Editora.

Vanzella, A. (2018). *Sistema de detecção de queda e monitoramento da frequência cardíaca utilizando ESP8266 e protocolo MQTT*. PhD thesis, Universidade Estadual do Oeste do Parana - UNIOESTE, Foz do Iguaçu.

Weiser, M. (1991). The computer for the 21st century. In Baecker, R. M., Grudin, J., Buxton, W. A. S., and Greenberg, S., editors, *Human-computer interaction*, pages 933–940. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Zamberlan, A., Perozzo, R., Kurtz, G., Librelotto, G., and Fagan, S. (2014). Integrando agentes AgentSpeak(L) em ambientes pervasivos educacionais. In *WESAAC - Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações*, pages 1–13, Porto Alegre. SBC.