UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL - CPGEI

RODRIGO NUNES OLIVEIRA

ASSISTÊNCIA À AUTONOMIA DOMICILIAR EMPREGANDO PARADIGMA ORIENTADO A NOTIFICAÇÕES

DISSERTAÇÃO DE MESTRADO

CURITIBA

2019

RODRIGO NUNES OLIVEIRA

ASSISTÊNCIA À AUTONOMIA DOMICILIAR EMPREGANDO PARADIGMA ORIENTADO A NOTIFICAÇÕES

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná – Área de Concentração: Engenharia de Computação

Orientador: Prof. Dr. Percy Nohama
Co-orientador: Prof. Dr. Jean Marcelo Simão

CURITIBA

2019

Dados Internacionais de Catalogação na Publicação

Oliveira, Rodrigo Nunes
Assistência à autonomia domiciliar empregando paradigma orientado a notificações [recurso eletrônico] / Rodrigo Nunes Oliveira. -- 2020.
1 arquivo texto (95 f.) : PDF ; 3,96 MB.

Modo de acesso: World Wide Web
Título extraído da tela de título (visualizado em 4 fev. 2020)
Texto em português com resumo em inglês
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Curitiba, 2019
Bibliografia: f. 89-95.

1. Paradigma orientado a notificações. 2. Internet das coisas. 3. Equipamentos de autoajuda para deficientes. 4. Raspberry Pi. 5. Aplicativos móveis. 6. Dispositivos de treinamento simulado. 7. Automação residencial. 8. Idosos - Habitações. 9. Detectores. I. Nohama, Percy. II. Simão, Jean Marcelo. III. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial. IV. Título.

CDD: ed. 23 – 621.3

Biblioteca Central da UTFPR, Câmpus Curitiba
Bibliotecário: Adriano Lopes, CRB-9/1429

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Pesquisa e Pós-Graduação

TERMO DE APROVAÇÃO DE DISSERTAÇÃO Nº 842

A Dissertação de Mestrado intitulada **"Assistência à autonomia domiciliar empregando paradigma orientado a notificações"** defendida em sessão pública pelo candidato **Rodrigo Nunes Oliveira**, no dia **31 de outubro de 2019**, foi julgada para a obtenção do título de Mestre em Ciências, área de concentração **Engenharia de Computação**, e aprovada em sua forma final, pelo Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial.

BANCA EXAMINADORA:

Prof. Dr. Percy Nohama - Presidente – (UTFPR)
Prof. Dr. Cesar Augusto Tacla - (UTFPR)
Prof. Dr. Anderson Luis Szejka - (PUC-PR)

A via original deste documento encontra-se arquivada na Secretaria do Programa, contendo a assinatura da Coordenação após a entrega da versão corrigida do trabalho.

Curitiba, 31 de outubro de 2019.

DEDICATÓRIA

Dedico este trabalho primeiramente à Deus e a Jesus Cristo, Senhor e Salvador de minha vida, e a minha mãe Nair Musial Oliveira, que durante minha graduação, me esperou todas as noites com uma refeição quente, palavras de incentivo e com muito amor e carinho.

AGRADECIMENTOS

Inicialmente, agradeço a Deus pelo dom da vida, Seu plano soberano em minha vida e por estar comigo em todos os momentos em que pensei em desistir, concedendo-me ânimo e capacitação para a conclusão deste trabalho.

Agradeço a minha esposa Marilia e meus filhos Júlia, Davi e Laura, que foram privados por vários momentos de minha companhia e de minha atenção, e mesmo assim incentivaram-me, mesmo que indiretamente, a concluir este trabalho.

Aos meus orientadores Prof. Dr. Percy Nohama e Prof. Dr. Jean Marcelo Simão, que me instruíram, colaboraram e dedicaram seu precioso tempo direcionando-me e revisando este trabalho.

À toda a equipe de pesquisa em torno do Paradigma Orientado a Notificações (PON), que de forma direta ou indireta contribuíram para a conclusão da presente dissertação.

RESUMO

OLIVEIRA, Rodrigo Nunes. **Assistência à Autonomia Domiciliar Empregando Paradigma Orientado a Notificações.** 2019. 95 f. Dissertação de mestrado. Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI). Universidade Tecnológica Federal do Paraná (UTFPR). Curitiba, 2019.

O aumento da expectativa de vida da população reflete-se em crescimento na população de idosos, e inúmeras ações tornam-se necessárias para garantir ao idoso a qualidade de vida adequada, além de independência na realização de tarefas cotidianas. Dentre as aplicações da tecnologia da Internet das Coisas (IoT) encontra-se a criação de ambientes inteligentes com dispositivos interativos que permitem a integração entre o ambiente físico e o indivíduo. Uma aplicação viável dos ambientes inteligentes é a assistência à autonomia no domicílio (*Ambient Assisted Living* - AAL), que possibilita que idosos e pessoas com algum tipo de limitação recebam assistência em seu dia a dia, de forma independente e segura. Neste contexto, a fim de propiciar uma tecnologia assistiva, esta pesquisa objetiva a criação de um sistema AAL que possa ser extensível a novos cenários e que utilize de forma natural os recursos de paralelismo, processamento distribuído e modelagem de negócio por meio de regras de ambientes. Para isso, optou-se pelo uso do Paradigma Orientado a Notificações (PON). Assim, para implementação do sistema AAL aplicando PON, emprega-se microcomputadores Raspberry Pi, que são responsáveis pela comunicação com os sensores e atuadores físicos do ambiente, e por controlar notificações de forma distribuída entre os vários controladores presentes em uma mesma residência. Também foi necessária a criação de outros três sistemas: um portal web, responsável pela integração de todas as partes da solução, além do gerenciamento e da persistência de toda a configuração do ambiente e suas respectivas regras; um aplicativo para celulares para o monitoramento e controle do ambiente e indivíduos nele inseridos; e, por último, um simulador de ambientes que simula as regras e sensores em um ambiente virtual. O resultado da implementação do sistema foi avaliado de forma qualitativa, no tocante do atendimento do PON aos requisitos de sistemas AAL; e de forma quantitativa, no desempenho da solução em atividades de processamento como a leitura dos sensores, notificações internas do PON e notificações de rede. Com a utilização do PON, foi possível criar uma solução AAL extensível, pois permite vínculos de novos sensores e reconfiguração de suas regras em tempo real, além de monitorar o ambiente com o uso constante de paralelismo e do processamento distribuído.

Palavras-chave: Paradigma Orientado a Notificações. Ambientes Inteligentes. Assistência à Autonomia no Domicílio. Idosos.

ABSTRACT

OLIVEIRA, Rodrigo Nunes. **Ambient-Assisted Living Tool Using Notification-Oriented Paradigm**. 2019. 95 f. Master of Science in Electrical Engineering and Industrial Informatics. Graduate Program in Electrical and Computer Engineering (CPGEI). Universidade Tecnológica Federal do Paraná (UTFPR). Curitiba, 2019.

The rising in life expectancy is reflected in an increase in the elderly population and numerous actions become necessary to ensure the elderly a proper quality of life and independence in performing daily tasks. One of the applications of the Internet of Things technology (IoT) is the creation of smart environments with interactive devices allowing integration between the physical environment and the individual. A viable application of intelligent environments is the Ambient Assisted Living (AAL), which can enable elderly and people with some kind of limitations be assisted in their daily routine, independently and safely. In this context, in order to provide assistive technology, this research aims to create an AAL system that can be extended to new scenarios and that use parallel resources naturally, distributed resources and business modeling through environment rules. To do so, was choose to use the Notification Oriented Paradigm (PON). Thus, for implementation of the AAL system applying PON, Raspberry Pi microcomputers are used, which are responsible for communicating with the physical sensors and actuators of the environment, and for controlling notifications distributed among the various controllers present in the same residence. Also, it had required the creation of three other systems: a website, responsible for the integration among all parts of the solution as well as the management and persistence of all the environments' configuration and their respective rules; a mobile app for monitoring and controlling the environment, and the people included in it; and lastly, a simulator that simulates rules and sensors in a virtual environment. The final system implementation was assessed qualitatively, by means of PON compliance analysis with AAL system requirements; and quantitatively by its performance in processing activities such as sensor reading, PON internal notifications and network notifications. Using PON, it was possible to create an extensible AAL solution, as it allows the inclusion of new sensors and reconfiguration of their rules in real time, as well as monitoring the environment through the constant use of parallelism and distributed processing.

Keywords: Notification Oriented Paradigm. Smart Environments. Ambient Assisted Living. Elderly.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE SIGLAS, ACRÔNIMOS E ABREVIATURAS

SIGLA	ORIGINAL	TRADUÇÃO
AAL	*Ambient Assisted Living*	Assistência à autonomia no domicílio
AmI	*Ambient Intelligence*	Ambientes Inteligentes
API	*Application Programming Interface*	Interface de Programação de Aplicativos
FBE	*Fact Base Element*	Elemento da Base de Fatos
GPIO	*General Purpose Input/Output*	Entradas e saídas de uso geral
HDMI	*High-Definition Multimedia Interface*	Interface Multimídia de Alta Resolução
HTTP	*Hypertext Transfer Protocol*	Protocolo de Transferência de Hipertexto
IA	Inteligência Artificial	*Artificial Intelligence*
IoT	*Internet of Things*	Internet das Coisas
IP	*Internet Protocol*	Protocolo de Internet
JSON	*Javascript Object Notation*	Notação de Objetos em javascript
LAN	*Local Area Network*	Rede de Área Local
LED	*Light-emitting Diode*	Diodo emissor de Luz
NOCS	*Notification Oriented Care System*	Sistema de cuidados orientado a notificações
NOP	*Notification Oriented Paradigm*	Paradigma Orientado a Notificações
PIR	*Passive Infrared*	Infravermelho passivo
PON	Paradigma Orientado a Notificações	*Notification Oriented Paradigm*
REST	*Representational state transfer*	Transferência Representacional de Estado
RFID	*Radio-frequency Identification*	Identificação por radiofrequência
SMS	*Short Message Service*	Serviço de mensagens
TCP	*Transmission Control Protocol*	Protocolo de Controle de Transmissão
UDP	*User Datagram Protocol*	Protocolo de Datagramas de Usuário
UML	*Unified Modeling Language*	Linguagem de Modelagem Unificada
URL	*Uniform Resource Locator*	Localizador Uniforme de Recursos
UX	*User Experience*	Experiência de Usuário

VHDL	*VHSIC Hardware Description Language*	Linguagem de Descrição de Hardware VHSIC
VHSIC	*Very-High-Speed Integrated Circuit*	Circuito Integrado de Velocidade muito alta

SUMÁRIO

1 INTRODUÇÃO

Os avanços na medicina têm ampliado a expectativa de vida da população. De acordo com as Nações Unidas, a população global com 60 anos ou mais, chegou a 962 milhões em 2017, mais que o dobro quando comparada a 1980 quando era de 382 milhões. Espera-se que o número de idosos continue aumentando e, em 2050, ter-se-á aproximadamente 2,1 bilhões de pessoas com 60 anos ou mais no mundo. Em 2030, o número de idosos será maior que o total de crianças abaixo dos 10 anos (1,41 bilhão e 1,35 bilhão, respectivamente) e, em 2050, projeções indicam que haverá mais pessoas com 60 anos ou mais que adolescentes e jovens entre 10 e 24 anos (2,1 bilhões contra 2,0 bilhões) (UNITED NATIONS, 2017).

Entretanto, o avanço da idade é, geralmente, responsável por limitações motoras e sensoriais (visuais, auditivas ou táteis), além do surgimento de afecções crônicas. Este cenário cria vários desafios para a sociedade, entre os quais, ampliação da assistência médico-hospitalar, elevação subsequente de gastos com saúde e escassez de cuidadores de idosos. Não obstante, o envelhecimento da população deve ser encarado como uma oportunidade para se viver mais e melhor. Para tanto, algumas ações são necessárias para garantir ao idoso a qualidade de vida adequada e sua independência na execução de atividades cotidianas. Dentre elas, desponta a autonomia do idoso em seu domicílio, proporcionada pela computação senciente. Esta se baseia no monitoramento de um ambiente por meio de sensores e tomadas de decisões de acordo com as mudanças nesse ambiente e voltadas ao bem-estar do ser humano, tornando o ambiente em algo inteligente (RAMOS, 2007).

Os chamados ambientes inteligentes, no âmbito da computação senciente, permitem a interação inteligente e natural entre o indivíduo e o ambiente físico (COOK e DAS, 2007). Uma das suas aplicações é justamente o desenvolvimento de sistemas de assistência à autonomia no domicílio (*Ambient Assisted Living* - AAL). O AAL tem por objetivo principal possibilitar que pessoas com algum tipo de limitação motora e/ou sensorial, sejam auxiliadas em sua rotina diária, propiciando um estilo de vida independente e seguro o mais duradouro possível, dentro de seus ambientes pessoais (RASHIDI e MIHAILIDIS, 2013).

Atualmente, já é consenso que os sistemas AAL devem ser entidades dinâmicas que possibilitem adições de novos sensores, interpretação de novos comportamentos e execução de novas regras conforme o perfil de cada pessoa presente em uma determinada situação e contexto (ACAMPORA, COOK, RASHIDI e VASILAKOS, 2013), sendo que isto exige

complexidade maior na codificação desses sistemas. Na verdade, este tipo de sistema demanda possibilidades de atualização pelo próprio usuários, incluindo administradores do sistema, por meio de codificação em altíssimo nível. Ademais, esse tipo de sistema é de computação inerentemente distribuída para alcançar seus objetivos (KATSIRI, 2010).

Apenas as duas características destacadas anteriormente já se confrontam com as técnicas usuais de desenvolvimento de sistemas que são, não raro, de codificação difícil e inerentemente sequenciais e acoplantes, dificultando paralelismo e distribuição em geral (BANASZEWSKI *et al*., 2007; SIMÃO e STADZISZ, 2008; 2009; SIMÃO *et al*., 2012; RONSZCKA, 2019). Assim, novas técnicas de computação surgem, tais como a orientação a atores e a orientação a agentes que, muito embora atenuem os problemas, não os resolvem efetivamente. Isso ocorre porque o problema reside nos paradigmas de programação usuais que regem tais técnicas, como o Paradigma Imperativo e o Paradigma Declarativo, que tendem à computação sequencial prolixa (desperdício de processamento) e acoplante (dificuldades de modularização, paralelismo e distribuição) (BANASZEWSKI *et al*., 2007; SIMÃO e STADZISZ, 2008; 2009; SIMÃO *et al*., 2012; RONSZCKA, 2019).

Como alternativa, surgiu o chamado Paradigma Orientado a Notificações – PON (ou, em inglês, *Notification Oriented Paradigm* – NOP), que minimiza as entropias dos atuais paradigmas por evitar prolixia e ser altamente desacoplante. O PON tem tais propriedades graças ao seu foco na execução do programa por meio de notificações entre as entidades, de forma pontual e seletiva, eliminando a necessidade de execução sequencial (SIMÃO, BANASZEWSKI, TACLA e STADZISZ, 2012; SIMÃO e STADZISZ, 2007, 2009). Aliado a isso, por programação orientada a regras, o PON proporciona a criação de aplicações distribuídas, consistentes e robustas de maneira simples. Estas características são requisitos em diversos sistemas de computação contemporâneos e salientam-se na computação senciente, particularmente, na AAL (SIMÃO, RENAUX, LINHARES e STADZISZ, 2014).

1.1 MOTIVAÇÃO

O aumento na expectativa de vida da população resulta em um acréscimo no número de residências com idosos que moram, ou passam a maior parte do dia, sozinhos. Dado o fato que 89% dos idosos preferem ficar no conforto de seus lares, ao invés de se mudarem para centros focados no cuidado de idosos, e levando-se em consideração os custos com enfermeiros em domicílio (CENTERS FOR DISEASE CONTROL AND PREVENTION,

2013), é fundamental o desenvolvimento de tecnologias que ajudem estes idosos nas tarefas do cotidiano em suas próprias casas (RASHIDI e MIHAILIDIS, 2013). Além de auxiliarem pessoas idosas, podem ser estendidas a pessoas com necessidades especiais, auxiliando-as em suas residências em tarefas cotidianas proporcionando segurança e independência (RASHIDI e MIHAILIDIS, 2013).

Dentre a aplicação dos ambientes inteligentes, destacam-se aquelas com foco no cuidado com a saúde das pessoas, como monitoramento constante, terapias e reabilitação, bem-estar emocional e persuasivo, hospitais inteligentes e assistência na autonomia no domicílio. Esta última chamada de *Ambient Assisted Living* (AAL), onde o ambiente inteligente auxilia pessoas com limitações em suas atividades cotidianas, na maioria dos casos idosos que residem sozinhos, proporcionando independência e qualidade de vida em seus lares (RASHIDI e MIHAILIDIS, 2013).

Os sistemas AAL precisam ser eficazes com tempo de resposta de processamento extremamente baixo, com possibilidade de execução de tarefas de forma paralela e com mecanismos de integração entre vários ambientes de uma mesma residência. Esses requisitos estão relacionados ao fato dos sistemas AAL também monitorarem condições de saúde do idoso, sendo que em situações que geram riscos para a pessoa, como identificação de uma queda, diminuição nos batimentos cardíacos ou presença de fogo em panelas, necessitam de processamento e decisão em segundos após o evento. Em caso de lentidão para identificar um risco e calcular as ações que devem ser tomadas, o momento em que essas ações forem executadas no ambiente pode ser tardio, pois o ambiente pode estar com outras situações, em decorrência da demora na tomada de decisão, ou o indivíduo pode ter sua condição de saúde agravada.

Sistemas AAL também possuem uma particularidade: precisam ter suas regras reconfiguradas de forma rápida e sem paradas no próprio sistema. Essa reconfiguração dinâmica das regras é necessária em decorrência da variação nas condições dos indivíduos, na ocorrência de eventos não cotidianos ou para atender tratamentos médicos temporários para o idoso (KATSIRI, 2010). No caso do idoso estar com um resfriado, as regras relacionadas à climatização precisam ser adaptadas à essa nova condição, podendo inclusive ser alteradas conforme a evolução do quadro clínico. Outro exemplo abrangeria situações de visitas à residência do usuário, onde os registros de movimentações pelos cômodos da casa, para identificação de padrões, precisam ser processados de forma separada, a fim de não identificar anomalias na rotina do residente.

1.1.1 COMPLEXIDADE NO DESENVOLVIMENTO DE SISTEMAS AAL

Para atender às necessidades de assistência domiciliar aos idosos, os sistemas AAL geralmente são aplicações complexas e que utilizam alguma inteligência artificial (RAMOS, 2007). Consequentemente, o desenvolvimento desses sistemas envolve complexidade na codificação, utilização de várias técnicas e tecnologias diferentes, como a execução paralela, o processamento distribuído e a modelagem baseada em regras (SIMÃO *et al.*, 2014).

Nos cenários onde o paradigma de programação, linguagem ou tecnologia utilizada, não abstraia parte da complexidade de codificação, há um esforço maior no desenvolvimento de sistemas AAL que assistam de forma mais abrangente as limitações individuais dos idosos, e que possam ser evoluídos juntamente com as novas tecnologias sensoriais. Neste contexto, essa seção apresenta as principais dificuldades de desenvolvimento de sistemas AAL, no que se refere à tecnologia (SIMÃO *et al.*, 2014).

O aumento no desempenho dos processadores que, de acordo com Moore (MOORE, 1965), dobraria a cada 24 meses, encontrou uma barreira no que diz respeito ao aumento da frequência de operação devido aos problemas de dissipação de potência (BORKAR e CHIEN, 2011). Este contexto favoreceu a criação de múltiplos núcleos no mesmo processador, sendo estes processadores chamados *multicores*, que possibilitam o processamento paralelo que, em tese, permitem aumentar o desempenho de execução computacional (RONSZCKA, 2019). Para o uso correto do paralelismo, os *softwares* dependem de técnicas adequadas de desenvolvimento, contudo as técnicas mais utilizadas atualmente apresentam problemas de acoplamento e redundância, além do modelo de programação ser naturalmente sequencial (RONSZCKA, 2019). Embora o desenvolvimento de *software* tenha evoluído para permitir alguma programação utilizando o paralelismo, tal evolução ainda não contempla de maneira efetiva o uso do paralelismo em ambientes *multicores* (RONSZCKA, 2019).

Dentre os requisitos implícitos de um sistema AAL, há necessidade de desenvolvimento de sistemas que utilizem de forma efetiva o paralelismo e a distribuição. O paralelismo ocorre em detrimento da necessidade da comunicação do sistema com os sensores e atuadores do ambiente, pois nos ambientes reais ela acontece de forma paralela, uma vez que temos sensores diferentes enviando suas leituras ao mesmo tempo para o controlador, e este, por sua vez, envia comandos para os atuadores do ambiente de forma

independente e também paralela, garantindo os baixos tempos de resposta na alteração do ambiente (SIMÃO e STADZISZ, 2007, 2009; BANASZEWSKI, 2009).

Com relação à distribuição, há a necessidade de comunicação entre os vários dispositivos que monitoram o ambiente; neste cenário, um processamento centralizado demandaria um aumento na latência de comunicação, uma dependência de um nó central e consequentemente uma maior probabilidade de falhas no sistema, pois caso o centralizador de processamento venha a falhar, todo o sistema falhará. Ao contrário, a programação distribuída permite que cada dispositivo controlador faça o processamento e tratamento das informações antes de submetê-la ao sistema central. As leituras da temperatura de um ambiente que ao chegar em determinada medição acionam a climatização deste ambiente, podem ser processadas e inferidas dentro de seu escopo (cômodo, por exemplo), e somente nos casos que seja necessário notificar outros ambientes, pode distribuir essa informação para os interessados (SIMÃO e STADZISZ, 2007, 2009; BANASZEWSKI, 2009; KATSIRI, 2010).

De forma geral, o chamado Paradigma Orientado a Notificações (PON) resolve, em certa medida, alguns dos problemas existentes nos paradigmas usuais de programação, pois seu mecanismo de execução previne a existência de redundâncias estruturais, ou seja, repetições de códigos, e redundâncias temporais, que são as reavaliações desnecessárias de código (RONSZCKA, 2019).

1.2 HIPÓTESE

A partir do cenário descrito, levantou-se, então, a seguinte hipótese de pesquisa: utilizando novos paradigmas de programação, é possível criar sistemas de assistência à autonomia no domicílio que utilizam os recursos de paralelismo, distribuição no processamento e modelagem do negócio em forma de regras.

1.3 OBJETIVOS DA PESQUISA

O objetivo principal desta pesquisa é a criação de um sistema AAL que possa ser extensível a novos cenários, ou seja, ambientes com novos sensores e atuadores, e que utilize de forma natural os recursos de paralelismo, processamento distribuído e modelagem do negócio por meio de regras de ambientes.

A fim de alcançar o objetivo principal, delineou-se um conjunto de objetivos específicos:

1) Desenvolver um sistema de sensoriamento múltiplo que permita a inclusão e a alteração de sensores e regras em tempo de execução de forma simplificada;
2) Desenvolver uma aplicação para dispositivos móveis que permita o monitoramento e a interação com o ambiente;
3) Desenvolver um simulador de ambientes que permita a avaliação de cenários distintos;
4) Avaliar o uso do Paradigma Orientado a Notificações no desenvolvimento do sistema AAL.

1.4 ORGANIZAÇÃO DO TRABALHO

Este documento está organizado em 8 capítulos, sendo o atual responsável pela, a motivação e contextualização do problema para elaboração deste trabalho assim como seus objetivos.

O capítulo 2 apresenta a revisão da literatura relacionada a este trabalho, onde inicialmente será abordado os conceitos de ambientes inteligentes, em seguida os sistemas de assistência à autonomia no domicílio (AAL) e, na sequência, os conceitos do Paradigma Orientado a Notificações e suas materializações em várias linguagens e tecnologias.

No capítulo 3, apresenta-se a metodologia utilizada na criação do aplicativo de assistência à autonomia no domicílio, enquanto no capítulo 0 é apresentado resultado da implementação da solução.

O capítulo 5 apresenta os resultados obtidos nos testes em ambientes simulados, o capítulo 6 a discussão sobres os resultados, e o capítulo 7 as conclusões do trabalho. Por fim, no capítulo 8 propõe-se futuros trabalhos para melhoria deste projeto e/ou sua ampliação.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são apresentados os principais conceitos que foram utilizados na elaboração desse trabalho, sendo primeiramente a contextualização dos ambientes inteligentes na seção 2.1. A seção 2.2 apresenta os conceitos dos ambientes de assistência à autonomia no domicílio (*Ambient Assisted Living)*. Por fim, a seção 2.3 contextualiza o PON e suas materializações.

2.1 AMBIENTES INTELIGENTES

A comissão europeia *Information Society Technologies Programme Advisory Group* (ISTAG) apresentou o conceito de ambientes inteligentes, em inglês *Ambient Intelligence* (AmI), em 2001. Devido ao fato do grande número de tecnologias envolvidas na concepção de ambientes inteligentes, existem vários trabalhos relacionados ao tema e que foram produzidos anteriormente à própria formalização do termo (ISTAG, 2001; 2002).

O próprio termo *Internet of Things* (IoT)*,* que se referencia a uma das tecnologias fundamentais para os ambientes inteligentes, foi criado em 1999 por Kevin Ashton, cofundador do *Auto-ID Center do Massachusetts Institute of Technology (MIT).* Ashton afirmou que a ideia original da IoT previa a conexão de todos os objetos físicos à Internet, com capacidade de capturar informações por meio de identificação por radiofrequência (RFID) e tecnologias de sensoriamento que permitia observar, identificar e compreender o mundo independentemente das pessoas e suas limitações de tempo, atenção e precisão (ASHTON, 2009).

Outros conceitos possuem algumas áreas que se sobrepõem aos ambientes inteligentes, como a computação ubíqua, computação pervasiva e a computação senciente (RAMOS, 2007). O conceito de computação ubíqua indica que há acesso a dispositivos computacionais em qualquer lugar, de forma integrada e coerente (WEISER, 1991).

Outro conceito visto comumente como sinônimo da computação ubíqua é a computação pervasiva. De acordo com Teresa Dillon, computação ubíqua é melhor considerada como estruturas subjacentes, sistemas, redes e dispositivos que são “invisíveis” e estão em todos lugares, permitindo que novos dispositivos e ferramentas se conectem de forma transparente (*Plug-and-Play*), enquanto a computação pervasiva está relacionada a

todas as partes físicas de nossas vidas, como o telefone celular, relógios inteligentes e sensores vestíveis (RAMOS, 2007).

O termo senciente, quando se refere a coisas vivas, está relacionado à capacidade de representar e entender o ambiente por meio das informações coletadas pelos canais sensoriais. Estas características inspiram a computação que se refere à capacidade de monitorar o ambiente, aprendendo com as informações coletadas e agindo de acordo com as necessidades momentâneas deste ambiente (SIMÃO, RENAUX, LINHARES e STADZISZ, 2014).

Na computação senciente, a ideia é prover sistemas que sejam adaptativos conforme os indivíduos presentes no ambiente, como, por exemplo, um sistema que controla um asilo onde os idosos têm suas particularidades e necessidades. Contudo, o sistema também deve ter a capacidade de se adaptar a um mesmo indivíduo uma vez que a condição de saúde pode ser alterada de um dia para o outro, e também ser extensível para suportar o ingresso de novos indivíduos no ambiente com suas próprias necessidades e limitações (OLIVEIRA, 2016; OLIVEIRA, *et al.*, 2018).

A funcionalidade de um ambiente inteligente pode ser descrita como um ciclo composto por uma etapa de monitoramento do estado do ambiente, seguido pelo seu processamento para alcançar um objetivo específico ou antecipar respostas a possíveis ações, até a atuação sobre o ambiente para alterar seu estado. O monitoramento ocorre por meio da coleta de dados sensoriais, gerados por dispositivos distribuídos em uma rede de sensores que fornece informações para o sistema tomar decisões de como alterar o ambiente (COOK e DAS, 2007).

Os componentes computacionais de um ambiente inteligente agem de forma distribuída e colaborativa em um determinado contexto (LOKE, 2006) como, por exemplo, sensores que monitoram uma determinada parte do ambiente e, de forma colaborativa, analisam e geram ações no ambiente monitorado (POTDAR, 2009).

Várias técnicas de Inteligência Artificial (IA) podem ser utilizadas em ambientes inteligentes, tais como representação do conhecimento, aprendizado de máquina, inteligência computacional, planejamento, reconhecimento de fala, linguagem natural, visão computacional, robótica e sistemas multiagentes. Essas técnicas de IA colaboram para uma análise mais complexa e uma ação mais precisa no ambiente (RAMOS, 2007).

Os ambientes inteligentes representam a visão de futuro da computação inteligente em que ambientes apoiam as pessoas que habitam neles. Nesse novo paradigma

computacional, os dados de entradas e saídas convencionais não existem mais; ao invés disso, os sensores e processadores estão integrados nos objetos do dia a dia, trabalhando juntos em harmonia para apoiar as pessoas. Ao confiar em várias técnicas de inteligência artificial, a AmI promete a interpretação bem-sucedida da riqueza de informações contextuais obtidas desses sensores incorporados e adaptará o ambiente às necessidades do usuário de maneira transparente e antecipada (ACAMPORA, COOK, RASHIDI e VASILAKOS, 2013).

Em relação à infraestrutura, os ambientes inteligentes voltados para cuidados com a saúde podem utilizar redes chamadas BAN (*Body Area Networks*) onde vários sensores são anexados nas roupas, no corpo ou até mesmo implantados sob a pele, permitindo um monitoramento constante das condições de saúde do indivíduo; e WMSN (*Wireless Mesh Sensor Network*) composto por sensores de ambiente que coletam vários tipos de dados para deduzir atividades dos habitantes e antecipar suas necessidades, aumentando o conforto e qualidade de vida. Nas redes WMSN, cada nó, além de capturar e distribuir seus próprios dados, serve como um repetidor para os outros nós, ou seja, cada sensor colabora para a propagação de dados na rede (ACAMPORA, COOK, RASHIDI e VASILAKOS, 2013).

O desenvolvimento de sistemas voltados para ambientes inteligentes pode ser composto da utilização de alguns algoritmos e métodos, como o reconhecimento de atividades, identificação de padrões comportamentais, detecção de anomalias, planejamento das atividades diárias, apoio à decisão, e anonimato e preservação da privacidade. E as aplicações desses sistemas podem ocorrer no monitoramento contínuo de saúde, monitoramento comportamental, monitoramento para detecção de emergências, assistência na autonomia no domicílio, terapia e reabilitação, aplicações persuasivas de bem-estar, bem-estar emocional ou hospitais inteligentes (ACAMPORA, COOK, RASHIDI e VASILAKOS, 2013). A Tabela 1 apresenta um resumo dos tipos de aplicações dos ambientes inteligentes com foco na saúde, suas descrições e metodologias utilizadas.

Um dos problemas gerados por sistemas adaptativos é a definição da dependência entre as alterações do ambiente, por exemplo a climatização de um ambiente pode não interessar às regras dos demais ambientes. Katsiri propõe uma solução para lidar com os diferentes níveis de fatos e regras do ambiente chamada de Eventos Abstratos (*Abstract Events*), onde a ideia principal é que fatos primitivos, como a mudança de uma pessoa de lugar, devem afetar apenas regras primitivas como aquelas que desejam saber se a pessoa mudou de lugar. Contudo, regras primitivas podem gerar algo chamado de fatos abstratos

que quando gerados criam um evento abstrato, que pode ser utilizado de várias formas como notificar regras de outros ambientes que tem interesse neste evento (KATSIRI, 2005).

Tabela 1 – Aplicação de ambientes inteligentes no cuidado da saúde.

Tipo de Aplicação	***Descrição***	***Metodologia***
Monitoramento contínuo da saúde	Uso de sensores para monitorar as condições fisiológicas do usuário (temperatura, batimentos cardíacos, etc.)	Reconhecimento de atividade
Monitoramento comportamental	Uso de sensores para monitorar o comportamento humano (uso da TV, tempo de sono, etc.)	Reconhecimento de atividade
Detecção de emergências	Uso de sensores para detecção de quedas e ferimentos.	Reconhecimento de atividade
Assistência na autonomia	Criação de um ambiente que apoie pacientes e idosos durante suas atividades diárias.	Apoio à decisão
Terapia e Reabilitação	Apoio às pessoas que necessitam serviço de reabilitação, por meios remotos ou autônomos.	Reconhecimento de atividade e apoio à decisão
Bem-estar persuasivo	Sistemas que visam mudar atitudes pessoais com o objetivo de motivá-la a um estilo de vida mais saudável.	Reconhecimento de atividade e apoio à decisão
Bem-estar emocional	Sistemas baseados em percepções neurológicas e psicológicas para analisar emoções e melhorar o bem-estar.	Reconhecimento de atividade
Hospitais inteligentes	Melhorar a comunicação entre as partes interessadas do hospital, por meio de tecnologias ubíquas.	Apoio à decisão

Fonte: Adaptado de ACAMPORA, COOK, RASHIDI e VASILAKOS, 2013.

2.2 ASSISTÊNCIA À AUTONOMIA NO DOMICÍLIO – *AMBIENT ASSITED LIVING* (AAL)

Os sistemas AAL, quando desenvolvidos como ambientes inteligentes, podem ser usados na prevenção, tratamento e melhora das condições de saúde dos idosos. Sua aplicação pode ocorrer em atividades de monitoramento das condições de saúde, notificações e controle

no uso de medicamentos, detecção de quedas, monitoramento de segurança, auxílio nas tarefas cotidianas, facilidade na comunicação do idoso com familiares e enfermeiros, e até mesmo na geração e acompanhamento de um diário para indivíduos com demência (RASHIDI e MIHAILIDIS, 2013).

Um dos pontos fundamentais na concepção de sistemas AAL está relacionado ao design da aplicação uma vez que a maioria dos usuários teriam alguma limitação, e.g. são idosos com dificuldades visuais e na coordenação motora. É importante projetar uma interface com botões e letras grandes e com o uso de figuras intuitivas, de forma que seja a mais simples e amigável possível para se operar (RASHIDI e MIHAILIDIS, 2013).

Os sensores vestíveis, que monitoram as condições de saúde do usuário, não devem criar nenhum tipo de desconforto ou limitação nos movimentos. Uma alternativa é a utilização de dispositivos que o idoso já esteja familiarizado; por exemplo, celulares, *tablets* ou relógios e joias inteligentes. Não obstante, é evidente a resistência da maioria dos idosos em relação à tecnologia, sendo necessária uma preocupação em desenvolver soluções que exerçam seu papel com a menor dependência de ações diretas desses usuários (RASHIDI e MIHAILIDIS, 2013).

Evidentemente, o sistema AAL jamais deve substituir completamente a necessidade de cuidado humano, pois, nesse caso, poderia resultar inclusive em isolamento do usuário (RASHIDI e MIHAILIDIS, 2013).

A Tabela 2 mostra um resumo de vários projetos que se destinam à vida assistida. Por exemplo, o projeto CASAS da Washington State University que fornece um ambiente assistencial não invasivo para pacientes com demência. O projeto “Aging in Place” da Universidade do Missouri, visa fornecer um modelo de cuidados de longo prazo para idosos em termos de saúde e de suporte. O Elite Care é uma instalação de vida assistida equipada com sensores para monitorar indicadores como tempo na cama, peso corporal e inquietação do sono. O projeto Aware Home, da Georgia Tech, emprega uma variedade de sensores, como sensores de piso inteligente, além de robôs assistentes para monitorar e ajudar idosos (RASHIDI e MIHAILIDIS, 2013).

Tabela 2 – Projetos para ambientes inteligentes assistidos.

Projeto	***Instituição***
Aging In Place	U. of Missouri
Aware Home	Geogria Tech
CareLab	Germany

CareNet (MIDAS)	U. of Wales, UK
CASAS	Washington State U.
DOMUS	U. of Sherbrooke
Elite Care	OHSU
ENABLE	Netherlands
Gator Tech	UF
HIS	Grenoble U., France
MavHome	U. of Texas at Arlington
Millennium Home	Brunel U.
ProSAFE	LAAS, France
SELF	ETL, Japan
Smart Medical Home	Rochester U.
Ubiquitous Home	UCG, Japan
WTH	JMITI, Japan

Fonte: Adaptado de RASHIDI e MIHAILIDIS, 2013.

Alguns ambientes assistidos podem ser compostos por robôs que auxiliam na independência de pessoas idosas e com limitações. Neste cenário temos a casa robótica inteligente “Intelligent Sweet Home”, desenvolvido em KAIST na Korea, que possui o controle de cama inteligente, cadeira de rodas inteligente e guincho robótico utilizado para transferência fácil do usuário entre a cama e a cadeira de rodas (KWANG-HYUN *et al.*, 2007).

Uma aplicação comum dos sistemas AAL é o monitoramento das atividades diárias dos idosos, com o controle da duração, frequência e sensação de bem-estar do usuário após cada atividade. Este monitoramento pode auxiliar na identificação de anomalias na rotina diária e em sugestões de atividades que proporcionam bem-estar individual (HAIGH, *et al.*, 2004; SEBESTYEN, STOICA e HANGAN, 2016; SURYADEVARA e MUKHOPADHYAY, 2014).

O monitoramento constante do padrão de movimentação do idoso em sua residência, e da leitura de sinais como temperatura do corpo e número de batimentos cardíacos por minuto, possibilita a detecção de episódios de Transtorno Depressivo Maior (TDM), chamado também de depressão. A rápida identificação da doença permite o envio de notificações à familiares, cuidadores ou médicos, que podem tomar medidas antecipada no tratamento da doença (EDWING, FERRUZCA e IVAN, 2015).

Em alguns cenários específicos, sistemas AAL podem monitorar apenas o ambiente, sem considerar o monitoramento constante do indivíduo nele inserido, para situações

especiais ou tratamento/prevenção de doenças específicas, como soluções que monitoram a condição do ar, analisando a temperatura, umidade, concentração de partículas, níveis de monóxido de carbono e de outros gases. A partir da análise da qualidade do ar, é possível o envio de alertas para cuidadores de idosos para os casos de residências que possuem pessoas com histórico de crise asmática (SILVA, *et al.*, 2015).

Em virtude do grande volume de dados de sensores trafegado nos sistemas AAL, e da necessidade de poder de processamento para desempenhar tarefas de análise dos dados, algumas soluções propõem que este processamento seja feito em ambiente *cloud* (nuvem), por permitir uma forma fácil e rápida de escalonamento de recursos computacionais. Contudo essa mudança do local de processamento implica em uma necessidade de comunicação constante e ininterrupta com nuvem, pois caso contrário o sistema pode ficar sem uma resposta do retorno do processamento (FORKAN, KHALIL e TARI, 2014).

As soluções AAL, tendem a focar em pontos específicos de assistência ao usuário, como monitoramentos para identificação de quedas ou episódios de depressão, apoio robótico para movimentação dentro da residência e análise da condição de saúde. Contudo, nesta dissertação, propõe-se um sistema que permita a inclusão de diversos sensores e uma configuração de alto nível, com o objetivo de tornar a aplicação extensível às mais variadas necessidades dos idosos.

2.3 PARADIGMA ORIENTADO A NOTIFICAÇÕES - PON

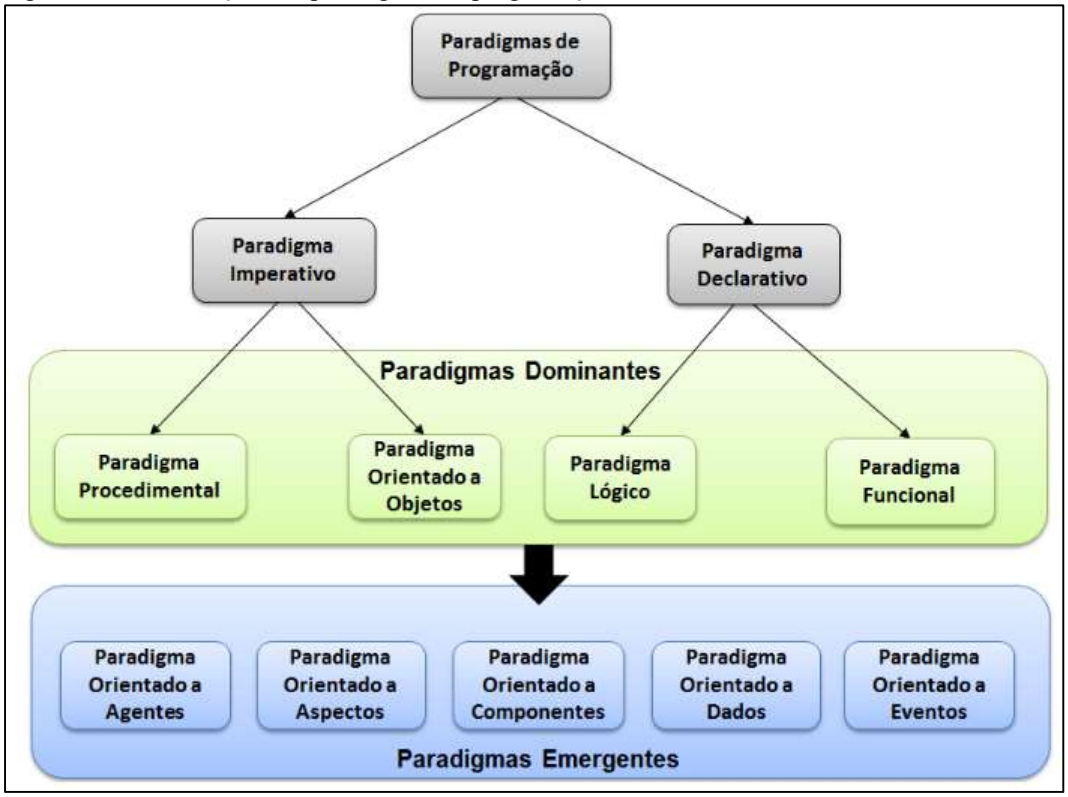
As linguagens de programação são regidas por paradigmas de programação, onde o nível de abstração de cada linguagem está vinculado às características de seus paradigmas. De forma resumida, pode-se considerar que há dois grandes paradigmas de programação, o Paradigma Imperativo (PI) e o Paradigma Declarativo (PD) (ROY e HARIDI, 2004; KAISLER, 2005; ROY, 2009; GABBRIELLI e MARTINI, 2010).

O PI pode ser dividido em outros dois paradigmas, o Paradigma Procedimental (PP) e o Paradigma Orientado a Objetos (OO), sendo que as linguagens de programação C e Pascal são exemplo de linguagens regidas pelo PP, enquanto as linguagens C# e Java são regidas pelo POO, havendo linguagens consideradas hibridas como o C++ (RONSZCKA, 2019).

O PD, por sua vez, pode ser dividido em outros dois paradigmas, o Paradigma Funcional (PF), que rege linguagens como LISP, e o Paradigma Lógico (PL), que rege linguagens como Prolog e OSP (ROY e HARIDI, 2004; KAISLER, 2005; ROY, 2009;

GABBRIELLI e MARTINI, 2010). Apesar da classificação de uma linguagem a um paradigma específico, é comum que estas linguagens tenham suas bases em mais de um paradigma de programação, inclusive linguagens atuais como Python, Swift e Lua utilizam de múltiplos paradigmas em sua estrutura (RONSZCKA, 2019).

Figura 1 – Classificação dos paradigmas de programação



Fonte: Ronszcka, 2012

A Figura 1 ilustra a classificação dos paradigmas de programação, com o destaque dos paradigmas dominantes, que orientam a construção das linguagens de programação vigentes, e os paradigmas emergentes, que em alguns casos, podem ser implementados a partir de diferentes paradigmas dominantes (RONSZCKA, 2019; BANASZEWSKI, 2009; HANSEN e FOSSUM, 2010; XAVIER *et al.*, 2014).

No PI, os *softwares* são construídos seguindo um modelo sequencial, baseado em buscas de elementos passivos, relacionando os dados (variáveis) com expressões lógico-causais (estruturas de decisão "se-senão"). Este modelo implica em redundâncias estruturais, ou seja, avaliações de uma mesma variável em várias partes do código; e em redundâncias temporais, que pode ser descrita como reavaliações de expressões que não sofreram alteração em laços de repetição (RONSZCKA, 2019; BANASZEWSKI *et al.*,2007; SIMÃO e

STADZISZ, 2008; GABBRIELLI e MARTINI, 2010; BROOKSHEAR, 2012; SIMÃO *et al*., 2012).

O PD, por sua vez, proporciona um nível maior de abstração e evitam muitas das redundâncias de execução, melhorando o desempenho das aplicações. Contudo, as soluções declarativas utilizam estrutura de dados, com alto custo computacional, causando sobrecarga de processamento. Mesmo com maior número de redundâncias as soluções baseadas no PI, normalmente apresentam melhor desempenho que àquelas baseadas em PD (RONSZCKA, 2019; BANASZEWSKI, 2009; SCOTT, 2016).

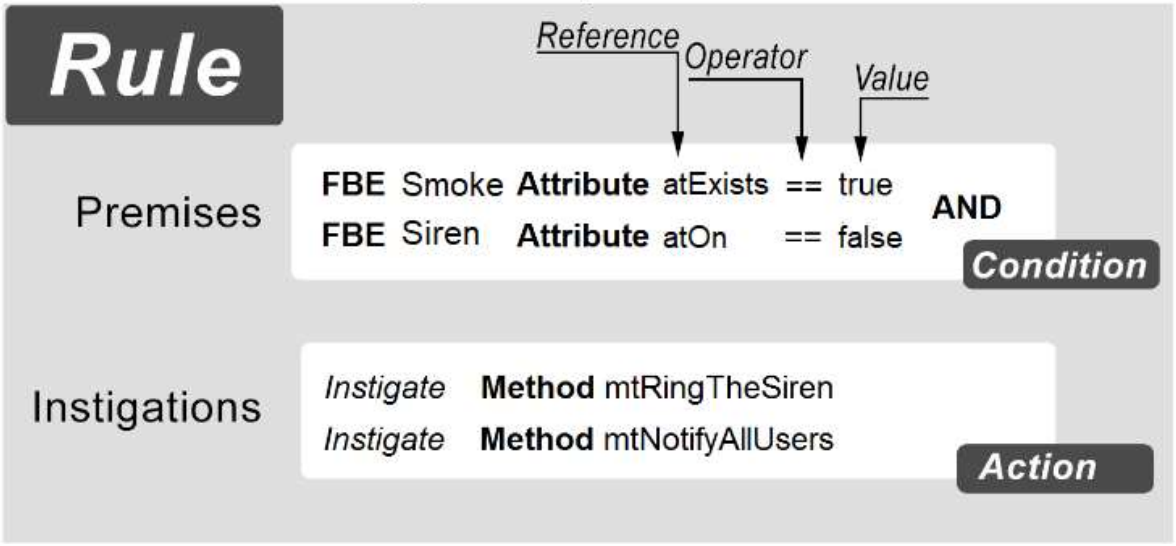
Uma técnica de programação denominada Paradigma Orientado a Notificações (PON) foi proposta por Simão (SIMÃO e STADZISZ, 2007, 2009), inicialmente, como uma solução de controle discreto de manufatura e, então, nova abordagem no sistema de inferência (SIMÃO, 2005), evoluindo, posteriormente, para um novo paradigma de programação.

A essência do PON está no seu sistema de inferência composto por entidades desacopladas que colaboram por meio de notificações precisas (SIMÃO e STADZISZ, 2007, 2009; SIMÃO, 2005). Isto resolve o problema da centralização e redundância causados pela atual abordagem de processamento lógico causal dos paradigmas atuais, problema esse que leva à subutilização da capacidade de processamento e acoplamento de código (SIMÃO e STADZISZ, 2007, 2009; BARRETO *et al.*, 2018).

O PON permite a construção de *softwares* melhores do ponto de vista de código, com performance apropriada e também distribuído (SIMÃO e STADZISZ, 2007, 2009). Sua estrutura utiliza corretamente os recursos de processamento melhorando o desempenho da aplicação, com a capacidade nativa da utilização de vários núcleos de processamento (*multicore*) e aplicações distribuídas de forma geral (RONSZCKA, 2019; 2012; LINHARES, SIMÃO e STADZISZ, 2015; BELMONTE, LINHARES, STADZISZ e SIMÃO, 2016; TALAU, 2016; SCHÜTZ, *et al.*, 2018; SCHÜTZ, 2019).

A Figura 2 apresenta um exemplo de *Rule* no PON, de forma semelhante a uma regra causal. A *Rule* é composta por *Condition* e por uma *Action*, sendo que a *Condition* é responsável pela decisão da *Rule*, enquanto a *Action* trata da execução das ações associadas (SIMÃO e STADZISZ, 2007, 2009; BANASZEWSKI, 2009; SIMÃO *et al.*, 2012).

Figura 2 – Exemplo de uma *Rule* que detecta fumaça no ambiente.



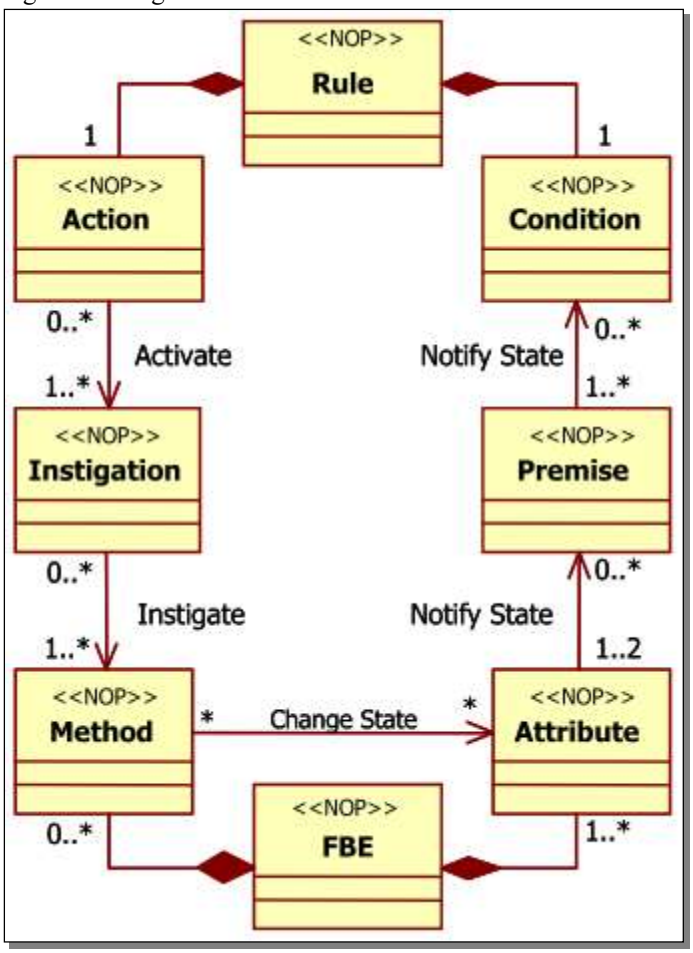
Fonte – Adaptado de RONCZCKA, 2019

A *Rule* representada na Figura 2 trata de uma situação de detecção de fumaça, (possível princípio de incêndio), em um ambiente inteligente. Nesta *Rule*, existem duas instâncias da *FBE (Fact-Base Element)*: o sensor de fumaça (*Smoke*) e a sirene (*Siren)*, que são as entidades do sistema, cada uma com seu respectivo atributo notificante *(Attribute)*. A *Condition* dessa *Rule* é composta por duas *Premises*, a primeira que verifica se existe presença de fumaça no ambiente e segunda que verifica se a sirene ainda não está ativa. Nas situações em que as duas *Premises* sejam verdadeiras, a *Action* da *Rule* é notificada, sendo esta *Action* composta por duas *Instigations* que executam os *Methods* responsáveis por acionar a sirene (*mtRingTheSiren)* e notificar os usuários do ambiente (*mtNotifyAllUsers)*.

A Figura 3 apresenta um diagrama de classes em UML, com as entidades do PON e seus relacionamentos. A *Condition* está vinculada a entidades relaciona à decisão da *Rule*, enquanto a *Action* está relacionada às entidades referentes à execução da *Rule* e dos elementos envolvidos (SIMÃO, RENAUX, LINHARES e STADZISZ, 2014).

Os elementos avaliados no PON são representados por uma entidade chamada *Fact Base Element* (FBE – Elemento da Base de Fatos), que é composta por um ou mais *Attributes* que possuem a capacidade de notificar as *Rules* interessadas quando seus valores sofrem uma mudança. Esta notificação ocorre em e por meio de uma entidade chamada *Premise*, sendo que apenas as *Premises* que desejam saber o atual valor do *Attribute* receberão a notificação da mudança de seu valor.

Figura 3 - Diagrama de classes do PON



Fonte: SIMÃO e STADZISZ, 2007; SIMÃO, RENAUX, LINHARES e STADZISZ, 2014

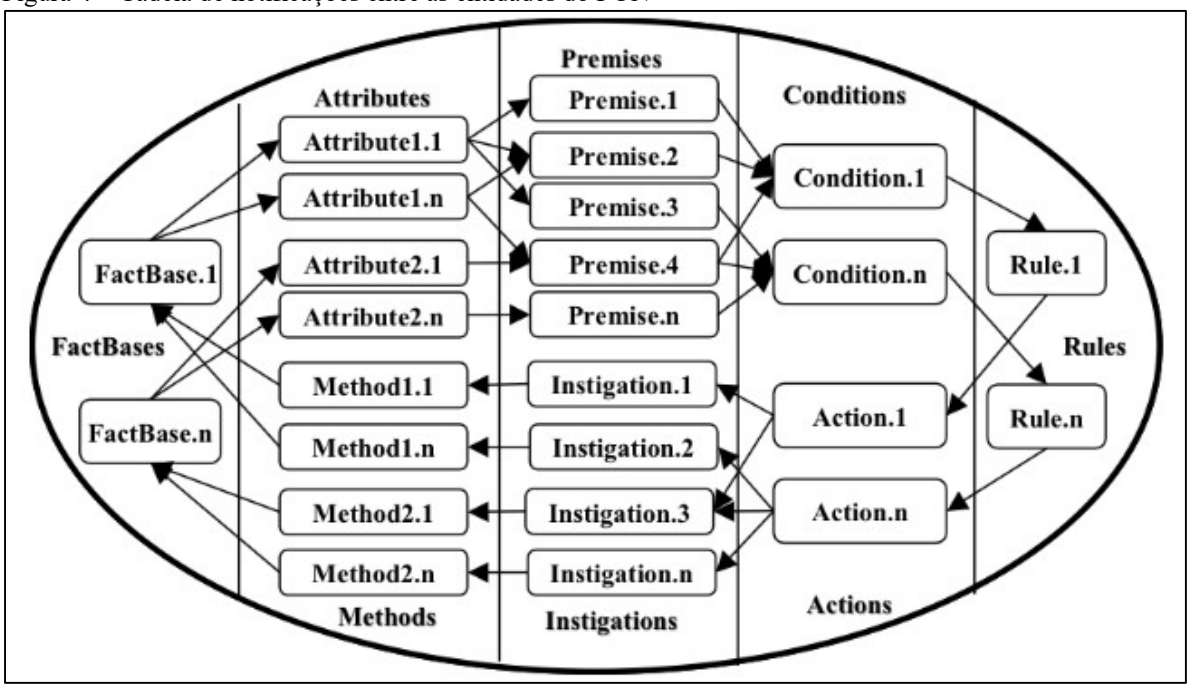
Para cada mudança de valor de um *Attribute* de um *FBE*, ocorrem notificações e avaliações lógicas somente nas *Premises* relacionadas. Assim que a *Premise* tem seu valor lógico alterado, ocorre uma subsequente notificação e avaliação lógica somente nas *Conditions* relacionadas com as eventuais mudanças no estado da *Premise*, (SIMÃO e STADZISZ, 2007; 2009; BANASZEWSKI, 2009; SIMÃO *et al.*, 2012).

De forma resumida, cada *Attribute* notifica as *Premises* relacionadas apenas quando há alteração em seu valor, e essa última por sua vez, notifica as *Condition*s relacionadas da mesma forma, apenas quando a *Premise* tem seu valor lógico alterado. No momento que a *Condition* é aprovada, a respectiva *Rule* é ativada, executando sua *Action*, que por meio de suas *Instigtions* inicia a execução dos *Methods*. Estes *Methods* podem alterar valores de *Attributes* iniciando todo o processo de inferência novamente (SIMÃO e STADZISZ, 2008; 2009; BANASZEWSKI, 2009; SIMÃO *et al.*, 2012).

Todo o processo de inferência do PON ocorre por intermédio de uma cadeia de notificações entre entidades computacionais, constituindo, dessa forma, a centralidade de

inovação do PON. A Figura 4 ilustra essa sequência de notificações entre as entidades que compõem o PON.

Figura 4 – Cadeia de notificações entre as entidades do PON



Fonte: Banaszewski, 2009

Além de resolver os problemas de desempenho, o PON é naturalmente aderente ao desenvolvimento de sistemas paralelos e distribuídos, devido ao seu baixo acoplamento de entidades. Em suma, não há diferença se a notificação entre as entidades está ocorrendo no mesmo computador ou em computadores que estão em redes diferentes. Nada impede que a entidade *FBE* esteja sendo executada em um computador e ao ter o valor de um *Attribute* alterado, este notifique a *Premise* que esteja em outro computador. Outra vantagem deste desacoplamento é a capacidade de as notificações ocorrerem em paralelo utilizando, assim, a estrutura de processadores com múltiplos núcleos comum nos computadores atuais (SIMÃO, RENAUX, LINHARES e STADZISZ, 2014).

As primeiras materializações do PON ocorreram na forma de um *framework*, desenvolvido sobre C++, no qual o desenvolvedor só precisa implementar as *FBEs*, com seus *Attributes* e *Methods*, e a criar as regras a partir de uma interface sem se preocupar sobre a criação de ligações para o processo de notificações entre as entidades do PON, que ocorre em segundo plano no âmbito do *framework* (SIMÃO, RENAUX, LINHARES e STADZISZ, 2014).

Atualmente, existem *frameworks* do PON em várias linguagens como: C++, Java, VHDL e também em C#, sem o foco na IoT, (HENZEN, 2015). Além deles, há também um compilador e uma linguagem própria para o PON, denominada LingPON (FERREIRA, 2015; SANTOS, 2017), que permite gerar código para uma das versões do framework PON, bem como para código mais de baixo nível.

No *Framework* PON C++ 2.0, foi proposto um conceito de impertinência de *Premises*, que consiste basicamente em um sequenciamento de *Premises*, de forma que é possível manter desabilitadas àquelas que recebem um grande número de notificações seguidas sem alterações em seu valor lógico (RONSZCKA, 2012). Este tema será melhor detalhado no capítulo 0, seção 4.1.5, tendo em vista que neste trabalho foi implementada uma evolução neste conceito.

Também foi proposta uma nova arquitetura de computadores baseada nos conceitos do PON chamada NOCA (*Notification Oriented Computer Architecture*) (LINHARES, SIMÃO e STADZISZ, 2015), a qual fornece a possibilidade da utilização do paralelismo de maneira mais adequada. Para os testes e avaliações da arquitetura NOCA foi desenvolvido um simulador para esta arquitetura, o NOCASim (PORDEUS, 2017).

Para exemplificar uma aplicação em PON, considere-se uma *Rule* do ambiente que liga a iluminação e aciona um aviso sonoro quando há uma pessoa nele, e sendo a distância dessa pessoa em relação à porta menor ou igual a 50 cm. Nesse caso, haverá uma *FBE* para o ambiente com *Attributes* para controle das pessoas presentes, da distância até a porta, da iluminação, e do acionamento da sirene (aviso sonoro).

Tal *Rule* é composta por uma *Condition* vinculada a duas *Premises*: a primeira para verificar se existe pessoa no ambiente, e a segunda para verificar se a distância da pessoa até a porta está dentro do intervalo desejado. Quando todas as *Premises* forem verdadeiras, a *Condition* ativará a execução da *Rule* através de sua *Action* que, neste exemplo, é composta por uma *Instigation* vinculada a um *Method* responsável por ligar a iluminação do ambiente e a outro *Method* para acionar o aviso sonoro.

O Código 1 apresenta o código dessa *Rule* em PON, na linguagem LingPON, que é uma das implementações do paradigma (FERREIRA, 2015).

Código 1 - Exemplo da *Rule* e da FBE em LingPON

```
1  fbe Ambiente
2      attributes
3          integer numPessoas 0;
4          integer distanciaPorta 0;
5          boolean lampada false;
6          boolean apito false;
7      end_attributes
8      methods
9          method AcendeLampada(lampada = true)
10         method AcionaApito(apito = true)
11     end_methods
12 end_fbe
13 inst
14     Ambiente ambiente
15 end_inst
16 rule Iluminacao
17     condition
18         subcondition Verifica
19         premise temPessoa ambiente.numPessoa > 0 and
20         premise pertoPorta ambiente.distanciaPorta <= 50
21         end_subcondition
22     end_condition
23     action
24         instigation Ilumina ambiente.AcendeLampada();
25         instigation AvisoSonoro ambiente.AcionaApito();
26     end_action
27 end_rule
```

Fonte: Autoria própria

A inovação apresentada pelo PON, com sua capacidade implícita de execuções em paralelo, processamento de forma distribuída, desacoplamento das entidades de outras partes de códigos e codificação em alto nível na forma de regras, tornam este paradigma extremamente atraente frente aos requisitos dos sistemas AAL. Requisitos estes, que podem ser atendidos de forma transparente nas soluções desenvolvidas sob o regimento do PON.

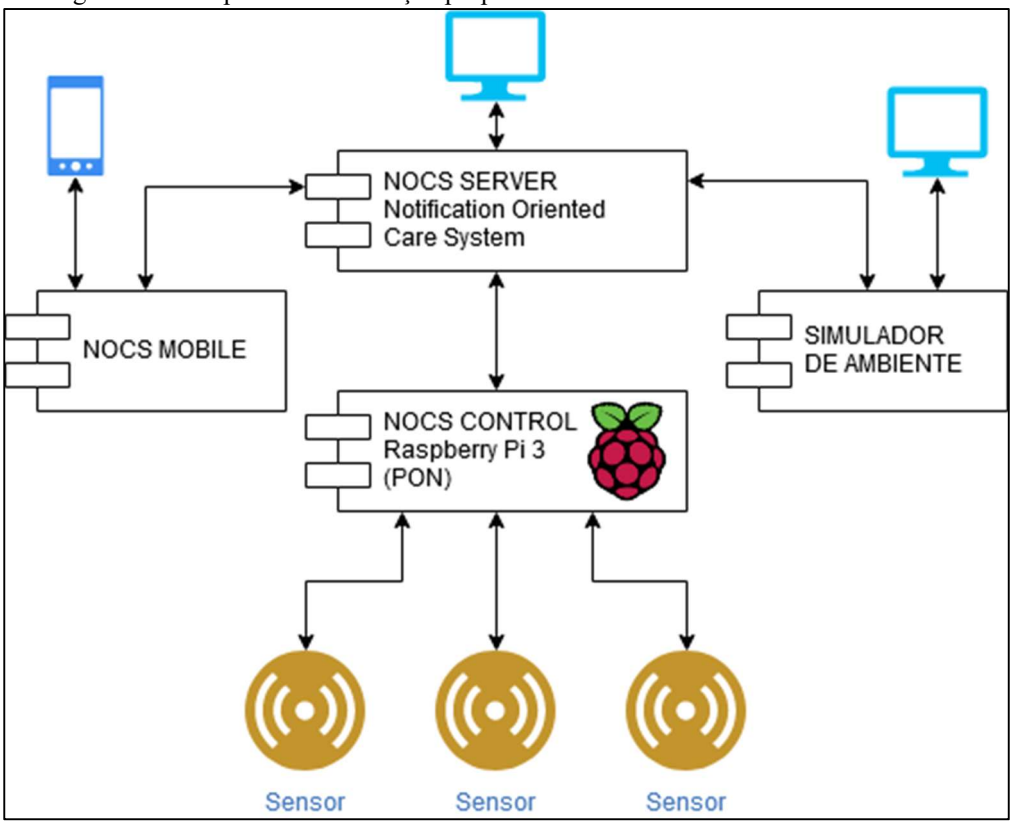
3 METODOLOGIA

Com o objetivo de desenvolver uma solução AAL, extensível, com instruções paralelizáveis, processamento distribuído e com uma modelagem utilizando regras, optou-se pelo desenvolvimento utilizando um paradigma de programação emergente. O Paradigma Orientado a Notificações (PON), no sistema que faz o controle do ambiente. Este controle iniciará por meio do recebimento das definições de todas as regras e sensores deste ambiente para iniciar o monitoramento.

Na sequência serão monitoradas as leituras dos sensores do ambiente, avaliando as condições de execução de uma "regra de ambiente", e, por consequência da aprovação da regra, o envio de ações que modificam este ambiente monitorado.

Outros sistemas que apoiam este monitoramento precisaram ser criados utilizando o paradigma orientado a objetos. Os sistemas assessoriais são responsáveis pela definição e persistência das regras de negócio que regem o controle de cada parte do ambiente; disponibilidade de visualização e alteração em tempo real no ambiente monitorado; e capacidade de simulação de cenários para validação das regras definidas para o ambiente.

Figura 5 – Diagrama de componentes da solução proposta



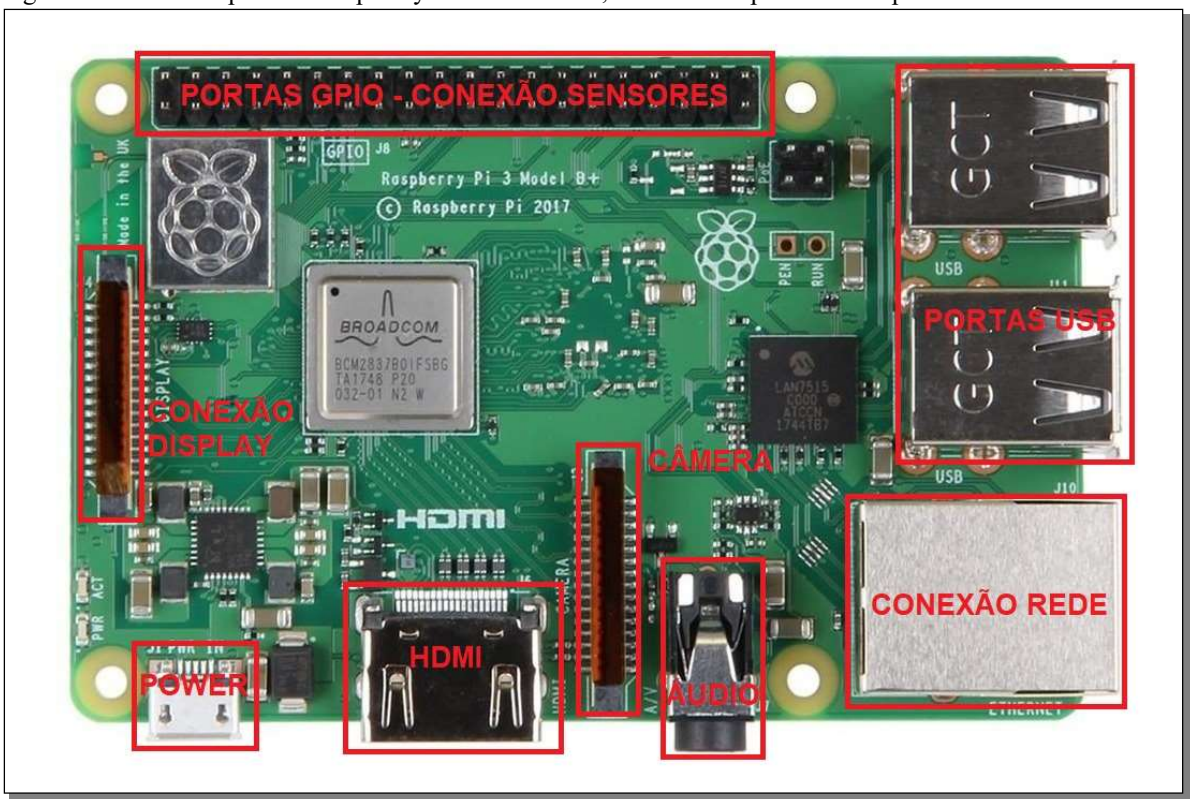
Fonte: Autoria própria

O diagrama apresentado na Figura 5 ilustra os componentes da solução proposta nesta dissertação, assim como as interações com seus respectivos dispositivos. Essa solução compõe-se por uma estrutura eletrônica de *hardware* programável por meio de um *software*, sendo o primeiro representado pelo microcomputador Raspberry Pi 3 e um conjunto de sensores e atuadores ligados a ele. O segundo composto por quatro aplicativos: o portal NOCS Server - *Notification Oriented Care System*, que pode ser traduzido como Sistema de Cuidado Orientado a Notificações, responsável pela centralização, gerenciamento e armazenamento da solução; o controle do Raspberry Pi 3, com seus sensores, denominado NOCS Control; um aplicativo para dispositivos móveis, o NOCS Mobile e um simulador de ambientes.

A Figura 5 também apresenta o fluxo de dados entre os componentes da solução NOCS. Primeiramente, há o portal NOCS Server que fornece as configurações de cada ambiente para os dispositivos Raspberry com sistema NOCS Control, sendo que cada Raspberry buscará as definições das *Rules* e sensores do seu respectivo ambiente. Para o monitoramento dos ambientes, há o NOCS Mobile que permite a visualização e alteração nos ambientes, a partir de consultas no NOCS Server. Por fim, para executar simulações de ambientes, existe um simulador que carrega as definições do ambiente para validação de *Rules* específicas, carregadas também do NOCS Server.

Na Figura 6 pode ser visualizado o microcomputador Raspberry Pi 3 com a marcação dos seus principais componentes: portas GPIO para conexão de sensores e atuadores, portas USB, porta de rede LAN, saída de áudio, conector para câmera, saída HDMI, conector de energia (*power*) e conexão *display* para telas *touch-screen*.

Figura 6 – Microcomputador Raspberry Pi 3 modelo B+, com seus respectivos componentes



Fonte: Autoria própria

Para que fosse possível a criação do aplicativo NOCS Control no PON, sendo este responsável pelas leituras dos sensores, avaliações para tomada de decisão e envios de ações para o ambiente quando necessários, primeiramente foi necessário materializar o PON em um *framework* voltado para a Internet das Coisas (IoT). O desenvolvimento do *framework*, implementado na linguagem C# e chamado de *Framework* PON.IoT C#, permite que o PON seja executado nos microcomputadores Raspberry Pi com sistema operacional Windows 10 IoT. Para a concepção do *Framework* PON.IoT C#, foram utilizados como referência a materialização *Framework* PON C++ 2.0 (RONSZCKA, 2012) e o *Framework* PON C# 1.0 (HENZEN, 2015).

Utilizando-se do *Framework* PON.IoT C#, foi possível criar o aplicativo NOCS Control, também em C#, que possui a capacidade de se comunicar com o NOCS Server, buscando as definições dos parâmetros do seu respectivo cômodo. A partir do carregamento das definições, o NOCS Control, instancia toda a estrutura do PON, com os sensores (*FBEs*) e as *Rules* com sua estrutura: *Conditions, Premises, Actions, Instigations* e *Methods*. O NOCS Control também inicia o processo de leitura dos sensores físicos e o envio de valores para os atuadores, ambos conectados no Raspberry Pi por meio das portas GPIO. Assim que há uma

alteração no valor de um sensor, o aplicativo envia este valor para a *FBE* iniciando todo o processo de inferência do PON.

Para o cadastro e definição dos sensores, ambientes e regras, foi desenvolvido o portal web NOCS Server, em ASP.NET C#, que notifica os microcomputadores Raspberry a cada alteração na parametrização do ambiente. O portal web permite o gerenciamento dos sensores, representados pela *FBE* do PON, gerenciamento das *Rules* com suas *Conditions* e *Premises* relacionadas, *Actions* e *Instigations* relacionadas, e, ainda, dos *Methods,* que permitem a alteração dos atuadores no ambiente. Além do gerenciamento, o portal NOCS Server permite o monitoramento do ambiente e das pessoas.

Para o monitoramento e controle diário do ambiente, foi criado o NOCS Mobile que é um aplicativo para dispositivos móveis, e foi desenvolvido em C# utilizando a plataforma Xamarin. Por meio do celular, é possível o controle total do ambiente, como acender uma lâmpada, monitorar as condições de saúde das pessoas e acompanhar notificações da própria plataforma.

Por fim, para a simulação de ambientes no qual é possível alterar valores de sensores e atuadores, foi desenvolvida uma aplicação C# Windows Forms, para carregar as definições de sensores *FBE* e *Rules* e alterar os parâmetros com a finalidade de avaliar os resultados das execuções das *Rules*.

Além disso, o sistema desenvolvido permite a execução e acompanhamento do ambiente simulado. A aplicação NOCS Control, executada nos microcomputadores, recebe a definição das *Rules* referentes ao seu ambiente e inicia o monitoramento dos sensores físicos e a alteração no ambiente conforme a execução das *Rules* definidas.

Em virtude da dificuldade de testes desse sistema em uma residência real, optou-se para a execução do sistema em simulações físicas e ambientes virtuais simulados, avaliando a execução das regras, leitura de sensores e alteração nos valores dos atuadores ligados nas portas GPIO do Raspberry Pi.

Dentre as dificuldades encontradas para a execução de testes em uma residência, pode-se citar a necessidade de instalações físicas como passagem de cabos, instalação de sensores, atuadores, microcomputadores; alterações na caixa de energia da residência; modificações do sistema elétrico e hidráulico, e alterações em portas e janelas. Não obstante, há o desconforto que essas alterações podem gerar no morador, que baseado nos requisitos do sistema, deve ser uma pessoa idosa e que preferencialmente viva sozinho.

Por questões de segurança da informação, não é possível alterar esses valores de forma manual no portal NOCS ou até mesmo no NOCS Mobile, exceto pelo próprio monitoramento e gerenciamento dos sensores do ambiente como, por exemplo, acender uma lâmpada ou abrir uma porta. Por intermédio do simulador do ambiente, é possível alterar o valor de qualquer sensor ou atuador, forçando assim a execução das *Rules* com a finalidade de avaliar o resultado de sua ação no ambiente.

Para os testes, por meio da simulação do sistema, foram elaborados alguns cenários, todos ligados a microcomputadores Raspberry Pi. Em um dos cenários foram utilizados sensores de presença por infravermelho (PIR HC-SR501), sensor de distância por ultrassom (HC-SR04), juntamente com um atuador LED e um atuador do tipo alto-falante (módulo *buzzer* 5V passivo). A *Rule* criada consiste em verificar a presença de pessoas no ambiente e a distância dessa pessoa em relação à porta, acendendo a lâmpada à LED e acionando um apito no alto-falante quando a condição da *Rule* é satisfatória.

Outros cenários foram simulados como por exemplo acender e apagar lâmpadas em vários ambientes (sala, cozinha, banheiro, etc.); ligar e desligar a TV, por meio de sensor de infravermelho; verificar abertura de portas e janelas por meio de sensores magnéticos; identificar chama no fogão com a utilização de sensores de fogo; monitoramento da temperatura do ambiente; e verificar e controlar fluxo de água em torneiras e chuveiros através de válvulas de passagem de água.

Na Tabela 3, exibe-se uma lista de sensores e atuadores com suas principais aplicações, e que podem ser utilizados no sistema de assistência à autonomia no domicílio NOCS. Os dispositivos são agrupados por cômodos (ambientes) e controlados por microcomputadores. Por exemplo, o Raspberry 1 é responsável pelo monitoramento e ações relacionadas aos eventos no ambiente “cozinha”. Contudo, existem situações em que o estado de um sensor pode disparar uma ação em outro ambiente. Neste caso, é imprescindível uma aplicação distribuída que permita comunicação entre os vários microcomputadores de forma transparente e em tempo real.

Tabela 3– Exemplo de sensores e atuadores com suas principais aplicações em sistema de assistênia à autonomia no domicílio.

Dispositivos	***Aplicação***
Infravermelho	Identificação de movimento
Chaves magnéticas	Abertura e fechamento de portas
Luzes	Controle da iluminação

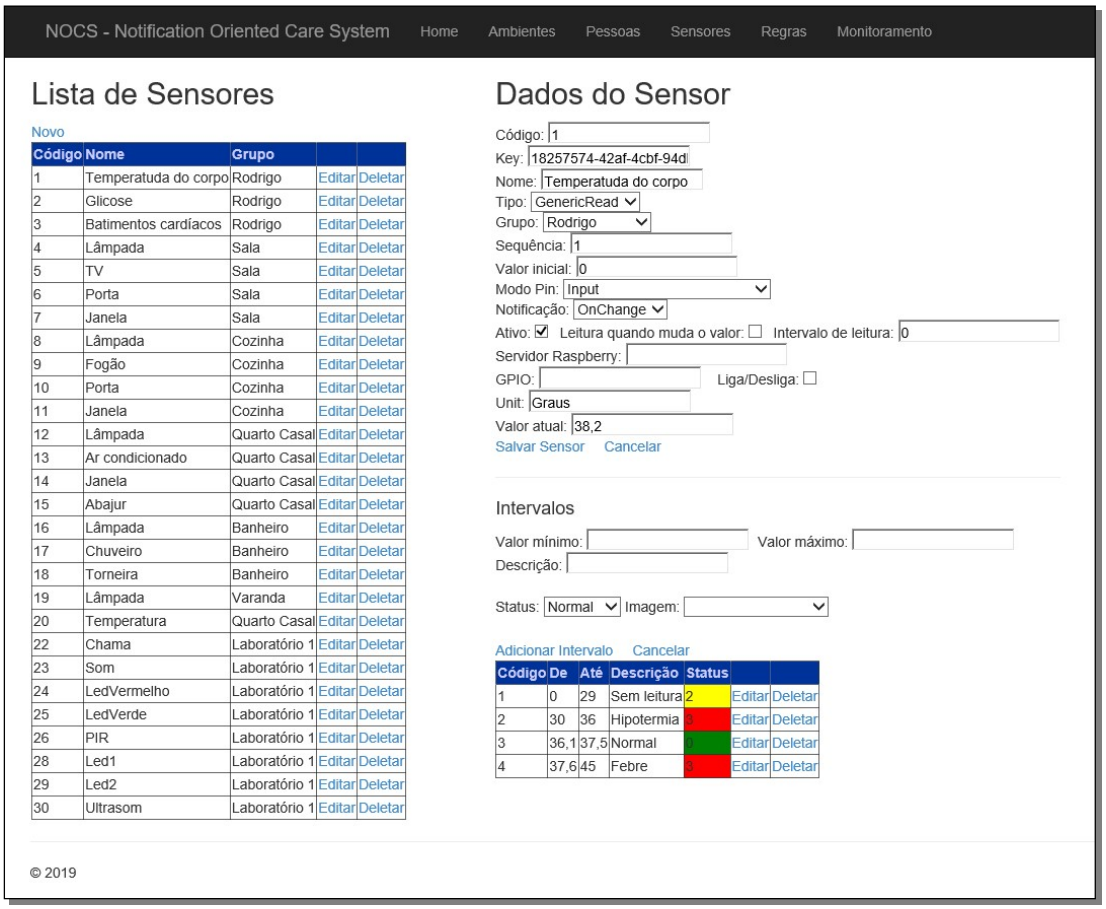
Ultrassom	Identificação de movimento
Microfone	Atividade no ambiente por meio do som
Giroscópio	Monitoramento da orientação
Glicosímetro	Monitoramento da glicose no sangue
Pressão	Monitoramento da pressão sanguínea
Eletrocardiograma	Monitoramento das atividades cardíacas
Oxímetro de pulso	Saturação do oxigênio no sangue
Térmico	Monitoramento da temperatura do corpo
Sensor de fumaça	Identificação de incêndios

Fonte: Adaptado de RASHIDI e MIHAILIDIS, 2013.

A Figura 7 apresenta a interface do portal NOCS para o gerenciamento dos sensores e atuadores onde é possível a sua inclusão e alteração, definindo informações como: tipo do sensor, grupo do sensor, ambiente que ele pertence – definido pelo IP ou nome do respectivo Raspberry, porta GPIO (*General Purpose Input/Output*) a qual o sensor está fisicamente ligado, e os intervalos dos valores que permitem definir rótulos para as faixas de leitura. Na Figura 7 pode-se ver, por exemplo, as faixas dos valores de um sensor de leitura da temperatura corporal, sendo de 0 a 29 ºC considerado como "Sem Leitura", de 30 a 36 ºC uma situação de Hipotermia, de 36,1 a 37,5 ºC rotulado como temperatura normal e de 37,6 a 45 ºC é considerado como uma situação de febre (hipertermia).

A criação de rótulos para as faixas de leituras permite uma simplificação na criação das *Rules*, pois uma regra pode ser modelada selecionado a faixa de leitura que deve disparar uma ação, por exemplo: caso a temperatura corporal esteja em febre, notificar familiares.

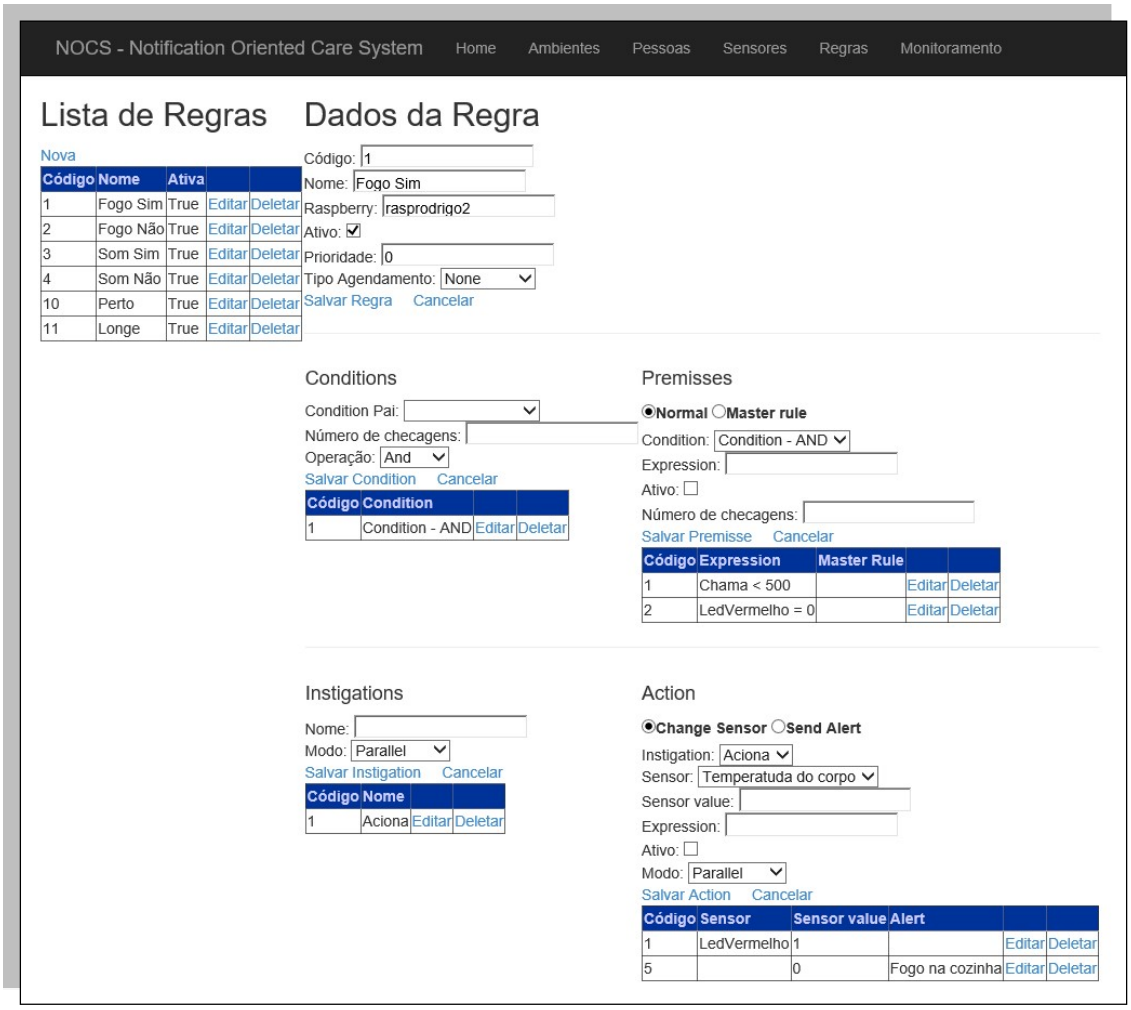
Figura 7 - Interface do portal NOCS para gerenciamento dinâmico de sensores e atuadores, além das faixas rotuladas das leituras destes sensores/atuadores



Fonte: Autoria própria

Na Figura 8, apresenta-se a interface do portal NOCS para gerenciamento dinâmico das *Rules*, onde é possível sua criação e alteração, além do gerenciamento das partes da *Rule* como as *Conditions*, *Premises*, *Instigations* e *Methods*. As *Conditions* de uma *Rule* são definidas por meio de uma ou mais *Premises* e o operador lógico entre elas - E (*And*) ou Ou (*Or*).

Figura 8 - Interface do portal NOCS para gerenciamento dinâmico de *Rules*, *Conditions*, *Premises*, *Instigations* e *Methods*



Fonte: Autoria própria

No gerenciamento de *Rules*, é possível criar *Subconditions* permitindo, assim, a formulação de várias *Premises* com operadores lógicos diferentes. O gerenciamento das *Premises* de uma *Condition* consiste na definição de uma expressão lógica que utiliza o valor de um ou mais sensores em sua formulação, por exemplo, Lâmpada = 1 (ligada), Temperatura < 10 (frio). A interface também permite a definição de um tipo especial de *Premise* que, ao invés de avaliar a situação dos sensores, analisa a aprovação de uma determinada *Rule*, por exemplo, se a *Rule* da iluminação já foi acionada.

As *Instigations* agrupam os *Methods* e podem ser executadas de forma sequencial ou paralela. No caso de um incêndio, acionado pela presença de fumaça no ambiente, ocorrerá, por exemplo, uma *Instigation* "fogo" que dispara o alarme e liga os chuveiros para apagar as chamas, mas poder-se-ia ter mais duas *Instigations*: uma para "abertura das portas" e "parada dos elevadores", e outra para o "desligamento da energia". Neste caso, o desligamento da energia deverá ser sequencial e não paralelo, para garantir que ao cortar a

energia os elevadores estejam parados e com as portas abertas; caso contrário, a energia poderia ser desligada antes que as pessoas saíssem dos elevadores.

Os *Methods* são responsáveis por envio de dados aos atuadores ou até mesmo outros sistemas. Nos *Methods*, define-se o atuador alvo e um valor ou uma expressão que calcule o valor a ser enviado para o atuador. De forma semelhante às *Instigations*, os *Methods* também podem ser executados de forma sequencial ou paralela, por exemplo, uma *Rule* que verifica a temperatura do ambiente e liga o ar-condicionado, além de fechar a porta e a janela do ambiente. Caso o ar-condicionado seja ligado antes do fechamento da porta, o sistema pode disparar outra *Rule* que desliga o ar-condicionado em situações nas quais o mesmo esteja ligado e a porta do ambiente esteja aberta. Neste caso, o ar-condicionado poderia não ser ligado se os *Methods* fossem disparados paralelamente.

Para a simulação da execução das *Rules* e monitoramento dos sensores, desenvolveu-se uma interface de execução do ambiente onde é possível simular valores nos sensores e analisar as ações tomadas no ambiente, inclusive nas situações em que o sensor de um ambiente pode afetar o estado de outro, sendo possível, assim, testar o requisito de distribuição nos sistemas AAL. Durante a execução da simulação do ambiente, é possível alterar as *Rules*, *Premises*, *Instigations* e *Methods*, por meio da interface de gerenciamento, e acompanhar essas alterações na simulação em tempo real, sem a necessidade de qualquer parada no sistema, testando, assim, a capacidade da gestão dinâmica dos sensores e *Rules* em um sistema em execução.

No portal NOCS Server foi criado um ambiente chamado Laboratório 1 que tem como finalidade testes diversos. A criação do laboratório é uma forma de criar *Rules* específicas para testes de sensores, monitorar a execução e verificar as ações disparadas por essas *Rules* no ambiente.

Na simulação do ambiente, foram utilizados os seguintes sensores/atuadores: termômetro, ar-condicionado, porta, janela, iluminação, presença de pessoa no ambiente, hora do dia, controle da TV, sensor de chama, controle de fluxo de água em encanamentos, sensor de fumaça, alarme de incêndio e chuveiros antichamas (*sprinklers*). Com esses sensores/atuadores, foram criadas *Rules* que controlam a temperatura do ambiente, a iluminação, abertura de portas e janelas, uso do fogão, utilização de torneiras e chuveiros, uso do aparelho de televisão e situações de incêndio. Um exemplo das *Premises* de uma dessas regras é o controle da iluminação do ambiente que acende as luzes se as mesmas estiverem apagadas, se houver alguém no ambiente e se o horário estiver compreendido entre

18 e 23 h. O Código 2 apresenta a estrutura das regras de climatização responsáveis pelo controle do ar-condicionado do ambiente.

Código 2 - Definição das regras que controlam a temperatura do ambiente

```
1   Regra Ligar o ar-condicionado
2       Se
3           Há pessoas no ambiente E
4           A temperatura do ambiente é maior que 23 graus E
5           O ar-condicionado está desligado
6       Então
7           Ligue o ar-condicionado
8
9   Regra Desligar o ar-condicionado
10      Se
11          O ar-condicionado está ligado E
12          Não há pessoas no ambiente
13      Então
14          Desligue o ar-condicionado
```

Fonte: Autoria própria

O resultado da simulação foi analisado de forma qualitativa, com o atendimento da aplicação frente aos requisitos dos sistemas AAL e de uma forma quantitativa, por meio de testes de *stress* e de notificações entre ambientes (distribuição). No teste de *stress*, utilizou-se várias simulações com 1, 5, 25, 50 e 100 ambientes, e em cada ambiente simulado, um total de onze sensores tiveram seus valores alterados de 0 a 30, de forma sequencial.

Ao término do teste, foram medidos o tempo total de processamento da simulação e contabilizados os números de notificações de sensores, análise de *Premises*, aprovações de *Rules* e execuções de *Methods* do ambiente, calculando, assim, as ações processadas em função do tempo, no microcomputador Raspberry Pi. No teste de notificações entre ambientes, foram enviadas 100 notificações de um microcomputador ao outro, ambos em uma mesma rede sem fio, utilizando os protocolos HTTP, TCP e UDP e, ao final do envio, foi medido o tempo total do teste (em milissegundos). Para cada protocolo, foi executado um total de dez testes para se obter a média de tempo de execução no envio das notificações.

Para analisar os ganhos que a aplicação teve com a implementação da impertinência dinâmica de *Premises* e *Conditions*, foi elaborado um teste com o tempo médio de execução de uma regra em cenários onde simula-se o número de vezes que uma *Premise* é notificada sem alterar sua situação, e em cada cenário desse é medido o tempo de execução normal e com a ativação da impertinência dinâmica, obtendo dessa forma uma comparação entre o desempenho na execução da regra com o recuso ativo e desabilitado.

Também foi executado um teste de *stress*, simulando 10 microcomputadores distintos buscando os parâmetros de seus ambientes, ao mesmo tempo e por várias vezes consecutivas, no portal web NOCS Server. Ao final do teste, mediu-se o tempo médio de resposta de cada requisição e a capacidade de requisições por minuto que o portal consegue responder.

A Tabela 4 apresenta, de forma resumida, as regras que foram utilizadas na avaliação do sistema AAL, tanto em simulações reais com o uso de sensores ligados ao Raspberry Pi, como nas simulações virtuais feitas pelo simulador de ambientes, com o uso do Raspberry Pi mas sem a utilização de sensores.

Tabela 4 – Lista de regras utilizadas no teste do sistema AAL proposto nesta pesquisa

Regras	***Descrição***
Temperatura do Corpo	Monitora, enviando alertas nas situações de febre ou hipotermia.
Nível de Glicose no sangue	Envio de alertas nos casos de hipoglicemia ou hiperglicemia.
Batimentos cardíacos/min.	Alerta de Bradicardia ou Taquicardia.
Acionamento de lâmpadas	Controle automático de iluminação baseado no horário do dia e na presença de pessoa no cômodo.
TV	Liga e desliga a TV, mudança de canal, conforme horário do dia.
Portas e Janelas	Monitoramento e controle de abertura e fechamento.
Uso do fogão	Detecção de chama, vazamento de GLP, risco de incêndio e de queimadura.
Climatização	Acionamento do ar-condicionado e controle da temperatura.
Acionamento do chuveiro	Controle e monitoramento do uso do chuveiro.
Acionamento de torneiras	Controle e monitoramento no consumo de água.
Qualidade do ar	Monitoramento da temperatura, umidade, nível de CO_2 e partículas no ar.
Uso de energia elétrica	Monitoramento do fluxo de energia elétrica em pontos específicos da residência.
Presença de pessoas na área externa	Regras de segurança contra possíveis invasões.
Inicio de incêndios	Monitoramento da presença de fumaça ou chamas, com acionamento de sirenes, e chuveiros de resfriamento.
Presença de pessoas nas áreas internas	Monitoramento da localização do idosos dentro da residência

Fonte: Autoria própria

4 RESULTADOS DA IMPLEMENTAÇÃO

Neste capítulo, apresenta-se a implementação da proposta conforme descrito no capítulo Metodologia, sendo que na seção 4.1 descreve-se a implementação referente à materialização do PON no *Framework* PON.IoT C#, enquanto a seção 4.2 descreve-se o aplicativo NOCS Control, que é executado nos microcomputadores Raspberry Pi responsáveis pelo controle dos sensores e atuadores de um ambiente.

Na sequência, a seção 4.3 apresenta a implementação do portal web NOCS Server responsável pelo gerenciamento dos sensores, pessoas e *Rules* dos ambientes, enquanto a seção 4.4 descreve a implementação do aplicativo para celulares NOCS Mobile, que tem como finalidade o monitoramento e controle diário de todos os ambientes do domicílio.

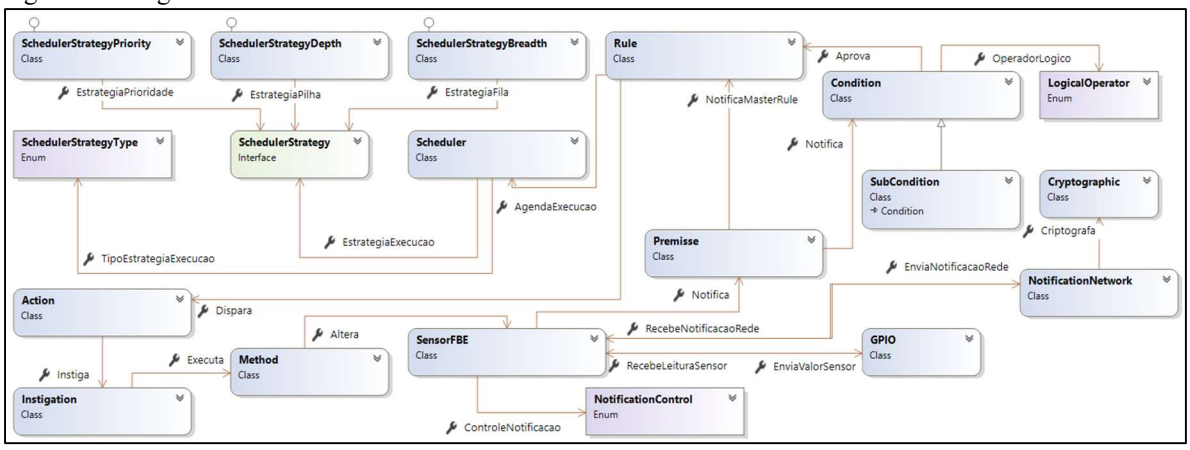
Por fim, na seção 4.5 apresenta-se a implementação do simulador de ambientes, utilizado na simulação do comportamento do sistema por meio da alteração dos valores de sensores e atuadores.

4.1 FRAMEWORK PON.IOT C#

Para a execução deste trabalho, foi necessário o desenvolvimento de uma nova materialização do PON: um *framework* em C# para execução deste paradigma em IoT, podendo ser executado em microcomputadores Raspberry Pi. Por meio da implementação deste trabalho, será possível avaliar recursos do próprio paradigma como paralelismo, distribuição e a capacidade de reconfiguração das regras em tempo de execução.

Para a execução do PON no Windows 10 IoT, rodando em dispositivos Raspberry Pi com sensores e atuadores conectados em suas portas de entradas e saídas de uso geral (GPIO), foi necessária a criação do *Framework* PON.IoT C#. A Figura 9 apresenta o diagrama de classes desse *framework*.

Figura 9 – Diagrama de classes do *Framework* PON.IoT C#



Fonte – Autoria própria

Na Tabela 5, apresentam-se os projetos envolvidos no *Framework* PON.IoT C#, e seu respectivo papel dentro da solução. Em suma, o *Framework* PON.IoT C# é composto pelo *core* do PON, onde ocorre todo o processo de instanciamento dos objetos notificantes e a inferência do PON; de classes de comunicação com os sensores e atuadores físicos ligados ao Raspberry; classes de notificações por meio da rede de computadores; e uma classe responsável pela criptografia e segurança dos dados trafegados na rede.

Tabela 5– Lista de projetos que compõem o Framework PON.IoT C#.

Projeto	***Descrição***
PON.IoT.Crypto	Biblioteca responsável pela criptografia das mensagens enviadas de um ambiente para outro.
PON.IoT.Library	Biblioteca PON para IoT que implementa todo o processo de inferência por notificação de forma encapsulada.
PON.IoT.NotificationClient	Classes para envio de notificações para outros ambientes.
PON.Iot.NotificationServer	Classes para recebimento de notificações de outros ambientes.
PON.IoT.GPIO.Input	Classes que efetuam a leitura dos sensores ligados ao Raspberry.
PON.IoT.GPIO.Output	Classes que enviam valores para os atuadores ligados ao Raspberry.

Fonte: Autoria própria.

Apesar desse desenvolvimento ter usado como referências o *Framework* PON C++ 2.0 e o *Framework* PON C# 1.0, foram necessárias várias alterações e novas implementações para que o PON atendesse à necessidade dos sistemas AAL. Melhorias como a criação de um método na *Rule* (*CheckApprove*()) que efetua uma verificação inicial de sua condição (aprovada ou reprovada) e a adoção do modificador de acesso *internal*, presente no C#, que restringe o acesso aos métodos *internal*, apenas pelas classes que pertencem ao mesmo

assembly. Desta forma, a biblioteca PON.IoT expõe apenas os métodos que precisam ser conhecidos pelo desenvolvedor na criação de uma aplicação PON, ocultando os métodos de uso interno para a execução de inferência por notificações do PON. Outras alterações são mais extensas e são descritas nas subseções seguintes.

4.1.1 *FBE* CONFORMADA PARA IOT

O primeiro passo foi aplicar uma unificação e simplificação nas entidades *FBE* e *Attributes*, pois no PON.IoT, em cada *FBE* tem-se apenas um atributo, relativo ao valor da leitura atual do sensor/atuador, que dispara avaliações das *Premises*. A nova *FBE* para IoT, chamada de *Sensor FBE*, também possui outros atributos, que não disparam notificações, e que são utilizados para definirem os parâmetros do sensor/atuador como o identificador único, nome, tipo, intervalo de leitura; a definição se o valor deve ser coletado a cada alteração; o modo de ligação e o número da porta conectada no Raspberry Pi; e ao se mudar seu valor, o mesmo deve ser enviado ao sensor/atuador.

Além das informações do sensor, a entidade *Sensor FBE* também possui atributos para controle da forma de notificação das *Premises*, como o controle de notificação, podendo ser *Always*, *On change* ou *Never*, a lista de dispositivos remotos que devem ser notificados, para os casos de notificação distribuída entre ambientes diferentes, além da própria lista de *Premises* que precisam ser notificadas, disparando, assim, todo o processo de inferência do PON. O Código 3 apresenta a implementação dos atributos da classe *Sensor FBE*.

Código 3 – Atributos da classe *Sensor FBE*

```
1  public class SensorFBE
2  {
3      private int ID;
4      private Double sensorValue;
5      public string SensorKey { get; private set; }
6      public string SensorName { get; set; }
7      public virtual string SensorType { get; set; }
8      public virtual int ReadingInterval { get; set; }
9      public virtual bool WhenValueChange { get; set; }
10     public virtual string PinMode { get; set; }
11     public virtual List<int> GpioPort { get; set; }
12     public bool WriteData { get; set; }
13     public NotificationControl NotificationControl { get; set; }
14     public List<string> NotifyServers { get; set; }
15     private List<Premisse> ListPremisse;
```

Fonte: Autoria própria.

Para atender ao requisito de paralelismo, tendo em vista que os processadores mais recentes possuem mais de um núcleo, permitindo execução de tarefas de forma paralela, todas as vezes que o objeto da classe *Sensor FBE* tiver o valor do sensor alterado, a notificação das *Premises* é disparada de forma paralela usando uma função própria do C#: Parallel.ForEach, que distribui cada iteração em um *core* do processador.

Em configurações onde o dispositivo possui núcleos em número maior ou igual ao número de *Premises*, cada *Premise* receberá sua notificação, avaliando a condição ao mesmo tempo que as demais *Premises* notificadas pelo *Sensor FBE* (que na prática é um *Attribute* pela fusão de *FBE* e *Attribute* já explicado). Quando há mais de um núcleo, mas não o suficiente para paralelizar todas as *Premises*, há concorrência entre as *Premises* em um mesmo núcleo e paralelismo em relação as *Premises* em outros núcleos.

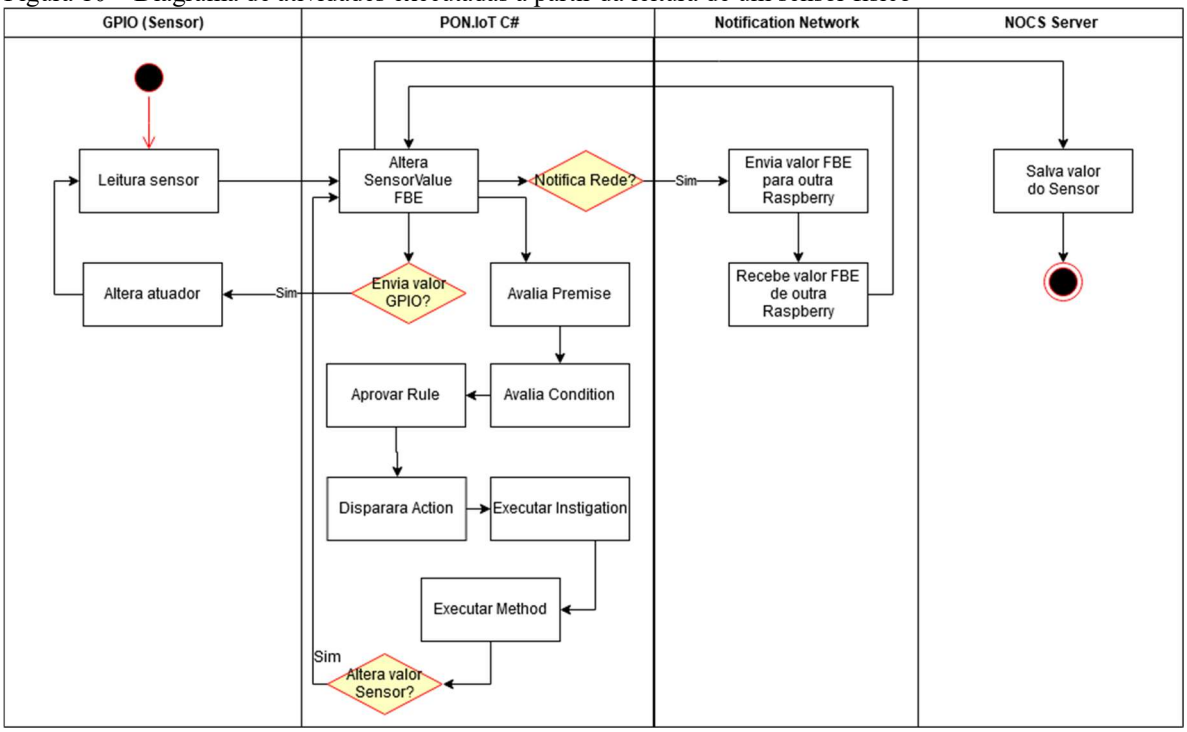
Em algumas situações, a *Premise* que deve ser notificada estará em outro ambiente, ou seja, sob o controle de outro dispositivo Raspberry Pi. Dessa forma, o objeto *Sensor FBE* envia uma notificação de forma concorrente ou paralela para objeto da classe responsável pelas notificações distribuídas *IoT.NotificationClient.Notification* que, por meio do seu método "*Send*", envia para os demais dispositivos via protocolo TCP a alteração no valor do *Sensor FBE*, além de enviar para o próprio portal NOCS Server a atualização do valor desse sensor, tornando o portal NOCS Server o centralizador das informações do ambiente em tempo real.

Ao disparar as notificações, o objeto da classe *Sensor FBE* também verifica se é necessário enviar o valor para a porta GPIO do Raspberry, isso para os casos onde há um atuador ligado a essa porta. A comunicação do *Framework* PON.IoT C# com as portas do Raspberry ocorre por meio de instância da classe IoT.GPIO*,* que por intermédio do seu método *Send* envia o valor para a porta vinculada ao atuador, podendo, desta forma, executar alterações no ambiente.

O preenchimento dos valores do sensor em instância ou objeto da classe *Sensor FBE*, quando o mesmo é proveniente da leitura de um sensor ligado no ambiente, ocorre por meio de objeto da classe IoT.GPIO.Input.ISensorGPIO, que monitora em intervalos parametrizados ou a cada alteração na leitura, enviando o valor coletado na porta GPIO do Raspberry para o atributo SensorValue de objeto pertinente da classe *Sensor FBE*, disparando novamente todo o processo de notificações. A Figura 10 apresenta um diagrama UML de atividades, onde é possível visualizar, a partir da leitura de um sensor físico, todo o processo de notificações entre os componentes do aplicativo NOCS Control (GPIO, *Framework*

PON.IoT C# e NotificationNetwork), até o envio do valor do sensor para persistência no portal NOCS Server.

Figura 10 – Diagrama de atividades executadas a partir da leitura de um sensor físico



Fonte – Autoria própria

Algumas *Rules* terão como resultado de sua execução a geração de um alerta, ou seja, a *Rule* não mudará nada no ambiente, não enviará alteração para sensores ou atuadores, apenas precisará registrar um alerta para ser acompanhado pelo indivíduo presente no ambiente ou para as pessoas que monitoram esse ambiente, familiares ou assistentes sociais, por exemplo. Para atender a esse requisito, foi adicionado ao *Framework* PON.IoT C# um tipo especial de *Method*, que ao invés de mudar o valor de um *Sensor FBE*, ele envia um alerta para o portal NOCS Server, que registra e apresenta esse alerta no aplicativo do celular e no próprio portal, além de possibilitar o envio desse alerta por e-mail, SMS, aplicativos de mensagens etc. Um exemplo de aplicação deste recurso seria nas situações em que um termômetro corpóreo do idoso registre uma situação de febre; neste caso, é importante enviar um alerta de febre para que seja ministrado um medicamento ou apenas para um acompanhamento nas alterações do quadro de saúde.

4.1.2 PARALELISMO

A aplicação do paralelismo no *Framework* PON.IoT C# não se restringe apenas às notificações das *Premises* e às notificações entre ambientes conforme já mencionados, o paralelismo via função Parallel.ForEach é utilizado na notificação das *Conditions* a partir de cada *Premise*, na notificação das *Instigations* a partir de cada *Action*, e na notificação (para a execução) dos *Methods*, executados por meio de cada *Instigation.*

Na verdade, *Instigations* e *Methods* podem ser disparados de forma paralela ou sequencial, sendo possível definir a forma da notificação na modelagem da *Rule*. Em algumas situações, o envio paralelo de toda a ação da *Rule* pode gerar resultado inesperados, pois há cenários em que uma ação deve ser executada obrigatoriamente depois de outra, por exemplo, nos casos de incêndios nos quais o desligamento da energia deverá ocorrer após a parada e abertura de todos elevadores.

O Código 4 apresenta a rotina de disparo dos *Methods* a partir da *Instigation*. As linhas 3 a 6 executam o disparo de forma paralela enquanto as linhas 7 a 10 percorrem a lista de *Methods* que devem ser disparados de forma sequencial, aguardando a conclusão de uma execução antes de iniciar a próxima.

Código 4 – Trecho de código da classe *Instigation,* com o método de execução dos *Methods* de forma paralela e sequencial.

```
1  internal void Execute()
2  {
3      Parallel.ForEach(ListMethod, (method) =>
4      {
5          method.Execute();
6      });
7      foreach (var method in ListMethodSequential)
8      {
9          method.Execute();
10     }
11 }
```

Fonte: Autoria própria.

4.1.3 NOTIFICAÇÕES DISTRIBUÍDAS

Em algumas situações, o valor de um *Sensor FBE* ou a execução de uma *Rule* de um ambiente deverá alterar valores e disparar avaliações de *Rules* em outros ambientes. Para que esse requisito fosse atendido, implementou-se classes de envio dos valores de sensores

por meio do protocolo TCP. Este protocolo foi escolhido após testes de notificações que compararam os protocolos: TCP, UDP e HTTP, sendo que apesar do protocolo UDP ter o menor tempo para a notificação, esse protocolo não garante a entrega do pacote, sendo essa limitação um item crítico em sistemas responsáveis por monitorarem ambientes e condições de saúde de pessoas. O resultado numérico do teste de notificação de rede com os três protocolos será apresentado no capítulo 5.

Cada objeto da classe *Sensor FBE* possui em seus atributos uma lista de microcomputadores (IPs ou nomes) que devem ser notificados. Quando o valor do sensor é alterado, um objeto pertinente à classe *Sensor FBE* chama o método "*Send*" de objeto pertinente à classe IoT.NotificationClient.Notification, passando um texto com três valores, separados pelo caractere barra vertical ou *pipe* (|), com o seguinte formato: N|SensorKey|SensorValue, onde o primeiro termo corresponde ao tipo da mensagem, podendo ser N para notificação de valores ou C para alterações (*Change*) em *Rules* ou *Sensor FBE*. O segundo termo da mensagem é a chave única do *Sensor FBE*, necessária para que o microcomputador que receber a mensagem, localize qual de seus sensores/atuadores deve ter o valor alterado. Por fim, o terceiro termo é o próprio valor do sensor.

Em um sistema que monitora e controla o ambiente, pensar em segurança da informação é essencial para garantir que as decisões tomadas sejam provenientes das próprias regras do ambiente. Por esse motivo, toda a troca de informação pela rede é criptografada, pois caso um sistema estranho tente interceptar, modificar ou até mesmo forçar uma alteração no valor de alguma mensagem enviada entre ambientes, não conseguirá interpretar a informação sem descriptografá-la, e da mesma forma não conseguirá enviar um comando válido pela rede para um ambiente.

O envio de notificação de rede ocorre de forma assíncrona na porta 9000. O Código 5 apresenta as linhas responsáveis pela abertura da comunicação, envio da mensagem e fechamento do canal de comunicação.

Código 5 – Trecho de código da classe NotificationNetwork, utilizado para enviar valores via TCP para outro microcomputador.

```
1  SocketClient _client = new SocketClient();
2  _client.OnDataReceive += Socket_OnDataRecive;
3  _client.OnError += Socket_OnError;
4  await _client.ConnectAsync(server, 9000); // Connect
5  await _client.SendAsync(String.Format("N|{0}|{1}", sensorKey, sensorValue));
6  await _client.DisconnectAsync(); // Disconnect
```

Fonte: Autoria própria.

O *Framework* PON.IoT C# permite que seja executado um método que estabelece uma "escuta" na porta 9000 para receber possíveis mensagens enviadas pela rede. Ao receber uma nova mensagem, a aplicação identifica se é uma mensagem de notificação ou se é uma mensagem de alteração das *Rules* ou *Sensor FBE*. Caso seja uma notificação, a mensagem é separada em termos, o *Sensor FBE* é localizado por meio da chave enviada na mensagem e o valor desse sensor é alterado conforme o valor enviado na notificação.

Por outro lado, se a mensagem for de alteração nas *Rules* ou *Sensor FBE* controlados por esse Raspberry, o sistema faz uma conexão via HTTP na API REST do portal NOCS Server e busca a nova definição das *Rules* e *Sensor FBEs*, promovendo assim uma atualização em seus parâmetros. O Código 6 apresenta o tratamento das mensagens recebidas, sendo que as linhas 4 a 10 são responsáveis pelo processamento do recebimento de mensagens com novos valores de sensores.

Código 6 – Trecho de código da classe NotificationNetwork, utilizado para tratar as mensagens recebidas via TCP de outros microcomputadores.

```
1  private static async void Server_OnDataReceive(ServerTcp sender, string args)
2  {
3      var msg = args.Split('|');
4      if (msg?.Count() > 1 && msg[1]?.Length > 0)
5      {
6          if (msg[0] == "N" && Repository.ListFBE != null) // Notification
7          {
8              var sensor = Repository.ListFBE.Where(a => a.SensorKey == msg[1]);
9              if (sensor != null)
10                 sensor.SensorValue    =    Convert.ToDouble(msg[2].Replace(".",
11         }
12         else if(msg[0] == "C") // Change Ex.: C|S1 (sensor 1) or C|R1 (Rule 1)
13         {
14             HttpClient client = new HttpClient();
15             client.BaseAddress = new Uri(Repository.ServerBase);
16             var resp = client.GetAsync("api/ws/GetRaspDefinition?IP=" +
17                 Repository.RaspName + "&Type=" + msg[1]).Result;
18             string json = await resp.Content.ReadAsStringAsync();
19             var sensorUpdates = Repository.SetChange(json);
20             if(sensorUpdates?.Count > 0)
21                 Listener.RemoveSensors(sensorUpdates);
22             Listener.Start();
23         }
24     }
25 }
```

Fonte: Autoria própria.

4.1.4 RECONFIGURAÇÃO DINÂMICA DO SISTEMA

Ao receber as definições de *Rules* e *Sensor FBEs* para um determinado ambiente, o *Framework* PON.IoT C# efetua uma varredura nessas configurações, criando todos os *Sensor FBEs*, e iniciando toda a estrutura de *Rules*, com suas *Premises*, *Conditions*, *Instigations* e *Methods*. Dessa forma, a aplicação inicia-se imediatamente após a montagem da estrutura de *Rules*, o monitoramento do ambiente e disparos de mudanças neste ambiente.

Um dos requisitos de sistemas AAL é a possibilidade de reconfiguração dinâmica e em tempo de execução das "regras de ambiente", pois como as definições dessas regras necessitam ser alteradas em situações não previstas, essa alteração não pode depender de uma parada total do monitoramento do ambiente ou até mesmo de uma atualização na aplicação dos microcomputadores. Um exemplo da necessidade de reconfiguração dinâmica do ambiente é quando o idoso, por exemplo, está com uma enfermidade que altere sua rotina. Neste caso, as 'regras de ambiente' no tocante à climatização, monitoramento do tempo de sono, uso do banheiro devem ser redefinidas para a nova condição de saúde, e que, nesse caso, é temporária.

Quando há uma mudança na definição de *Rules* ou *Sensor FBEs* no portal NOCS Server, é enviada uma mensagem via protocolo TCP para o(s) microcomputador(es) e ambiente(s) que serão afetados pela mudança. Essa mensagem é composta por dois termos separados pelo caractere barra vertical ou *pipe* (|), sendo o primeiro a letra C que representa alteração (*Change* em inglês), e o segundo termo composto de uma letra e um número onde a letra representa o que foi alterado, podendo ser R para *Rule* ou S para *Sensor FBE*, e o número é o identificador do objeto alterado. Como exemplos de mensagens enviadas nas alterações temos a mensagem C|R11, que pode ser traduzida como uma mudança na *Rule* número 11, e a mensagem C|S4, que corresponde a uma mudança no *Sensor FBE* número 4.

Ao receber a mensagem de alteração das definições, o *Framework* PON.IoT C# armazena a nova definição e instancia novamente, quando necessário, a estrutura atualizada das *Rules*, *Premises*, *Conditions*, *Instigations* e *Methods*, mantendo, assim, a aplicação em execução com as últimas parametrizações de monitoramento do ambiente. O Código 6 apresenta, da linha 12 a 22, a rotina que é executada quando o microcomputador recebe a mensagem de alteração de *Rules* e *Sensor FBEs*.

4.1.5 IMPERTINÊNCIA DINÂMICA DE PREMISES E SUBCONDITIONS

No *Framework* PON.IoT C# também foi implementado, de forma mais simples, a capacidade de aninhamento dito infinito de *Subconditions*, permitindo, assim, a elaboração de condições complexas nas regras onde cada grupo de *Subconditions* possui sua prioridade e seu operador lógico (*And* ou *Or*). Neste caso, a *Subcondition* pode ter um conjunto de *Premises*, um conjunto de outras *Subconditions* ou uma combinação de ambos.

Outrossim, o *Framework* PON C++ 2.0 apresentou uma nova funcionalidade chamada de Impertinência de *Premises*, na qual é possível "desabilitar" as notificações nas demais *Premises* de uma condição enquanto a primeira não tiver seu valor lógico como verdadeiro, cf. relatado na seção 2.3. Esta melhoria aumenta o desempenho da aplicação principalmente em cenários com grande número de notificações em condições que dependem da aprovação da primeira *Premise*.

No *Framework* PON.IoT C# essa melhoria foi estendida e chamada de Impertinência Dinâmica de *Premises*, ou seja, o desenvolvedor não precisa mais definir a impertinência de cada *Premise*, a própria aplicação controlará a ativação e desativação destas, baseando-se no resultado lógico da *Premise* anterior. Por exemplo, uma *Condition* que tem operador lógico *And* e que possui três *Premises*, será iniciada com a primeira habilitada e as demais desabilitadas, e no momento que a primeira for aprovada o sistema irá habilitar a segunda, e esta, por sua vez, habilitará a terceira em sua aprovação. Desta forma, o sistema melhoraria o número de notificações eliminando o consumo desnecessário de recursos.

O conceito de Impertinência Dinâmica de *Premises* foi estendido para as *Subconditions* criando a Impertinência Dinâmica de *Subconditions*. De forma análoga à execução das *Premises*, esta funcionalidade desabilita as demais *Subconditions* de uma condição e vai habilitando-as conforme a aprovação da *Subcondition* anterior. Lembrando que ao desabilitar uma *Subcondition*, o sistema está na verdade desabilitando todas as *Premises* e *Subconditions* desta condição, eliminando, consequentemente, as notificações desnecessárias.

Para se ter o melhor desempenho da aplicação com a utilização da Impertinência Dinâmica de *Premises* e *Subconditions*, o desenvolvedor deveria sequenciar estas estruturas considerando a com menor número de notificações para a maior. Contudo, os ambientes inteligentes são entidades vivas e em constante evolução, sendo que em um momento um sensor pode estar mudando de valor mais frequentemente que outros e, em outro momento,

este cenário se inverte. Pensando nesta limitação foi implementado no *Framework* PON.IoT C# o conceito de Sequenciamento Automático de *Premises* e *Subconditions*.

No Sequenciamento Automático de *Premises* e *Subconditions*, é possível definir um número máximo de notificações consecutivas que não afetam o valor da *Premise* ou *Subcondition*, e, baseando-se neste parâmetro, o sistema efetua um re-sequenciamento na *Condition*, habilitando a próxima *Premise* ou *Subcondition*, e desabilitando e enviando para o final da sequência a *Premise* ou *Subcondition* que esteja recebendo notificações de forma desnecessária. O Código 7 apresenta o método que executa a lógica de reordenação das *Premises* de uma *Condition*, esse método é chamado quando o número máximo de verificações consecutivas da *Premise* retornou falso. Desta forma, a *Premise* atual é desabilitada e a próxima *Premise* é habilitada para iniciar seu processo de verificação; caso a mesma já esteja com resultado de sua expressão como verdadeiro, a *Premise* seguinte é habilitada e, assim, sucessivamente.

Código 7 – Trecho de código da classe *Condition*, com o método que efetua a reordenação de *Premises*.

```
1  internal void ReorderPremisse(Premisse lastPremisse)
2  {
3      if (ListPremisses?.Count > 0 && ListPremisses.Last() != lastPremisse)
4      {
5          lastPremisse.Enable = false;
6          int index = ListPremisses.IndexOf(lastPremisse);
7          this.ListPremisses = this.ListPremisses.Where(a =>
8                                   a.ID != lastPremisse.ID).ToList();
9          this.ListPremisses.Add(lastPremisse);
10         var nextPremisse = this.ListPremisses[index];
11         nextPremisse.Enable = true;
12         if (nextPremisse.LogicValue)
13         {
14             IncrementElementsTrue();
15             EnablePremisse(nextPremisse);
16         }
17     }
18 }
```

Fonte: Autoria própria.

4.2 NOCS CONTROL

Com o intuito de desacoplar o *Framework* PON.IoT C# do sistema AAL proposto nessa dissertação, foi desenvolvido um aplicativo, utilizando-se do *Framework* PON.IoT C#, para tratar dos pontos específicos do sistema AAL, aplicativo este chamado de NOCS

Control. Dessa forma, o *Framework* PON.IoT C# poderá ser utilizado em outros projetos de IoT de forma independente e desacoplada.

O aplicativo NOCS Control é responsável pela busca dos parâmetros e configurações de ambientes no portal NOCS Server, e pela criação de toda a estrutura do PON nos microcomputadores Raspberry Pi. A Tabela 6 apresenta a lista de projetos que compõem o NOCS Control.

Tabela 6 – Lista de projetos que compõem o aplicativo NOCS Control.

Projeto	***Descrição***
PON.IoT.AALHeadless	Projeto inicial que é executado ao iniciar o sistema operacional do Raspberry.
PON.IoT.Entities	Classes para persistência dos *Sensor FBEs* e *Rules* em banco de dados.
PON.IoT.RepositorySensors	Repositório das *Rules* e *Sensor FBEs* para permitir alterações na aplicação em tempo de execução.
PON.IoT.Init	Biblioteca responsável pelo processamento inicial, buscando as definições do ambiente no portal NOCS Server, criando as *Rules* e monitoramento dos sensores físicos.
PON.IoT Library	*Framework* PON.IoT C# para o processo de inferência do PON.

Fonte: Autoria própria.

Para apresentar a capacidade de leitura e envio de dados para os sensores e atuadores ligados ao Raspberry, primeiramente é necessário explicar o processo de inicialização do sistema assim como a leitura das *Rules* definidas para o ambiente.

A aplicação NOCS Control que é executada no Raspberry é uma aplicação *Headless*, ou seja, não possui interface visual para o usuário e pode ser configurada para iniciar junto com o sistema operacional Windows 10 IoT. Dessa maneira, caso o microcomputador desligue, assim que for ligado novamente o sistema volta a funcionar automaticamente.

Ao iniciar a aplicação no Raspberry Pi, a primeira tarefa a ser executada pelo NOCS Control é a identificação do nome e IP do dispositivo que está sendo utilizado. Com essas informações, é possível executar o próximo passo que é a busca da definição das *Rules* e *Sensor FBEs* no portal NOCS Server, por meio de uma API REST, passando como parâmetro nessa busca o nome e IP do dispositivo. O servidor efetua a busca das definições de *Rules* e *Sensor FBEs* do dispositivo passado e retorna no formato JSON o resultado dessa pesquisa.

No momento que a aplicação NOCS Control recebe a definição dos *Sensor FBEs* e *Rules*, sejam elas no início da execução do programa ou após sua inicialização em situações

que as *Rules* ou *Sensor FBEs* foram alterados no servidor, o sistema inicia a criação de todos os sensores físicos, objetos da classe *Sensor FBE*, e todas as *Rules* com suas *Premises*, *Conditions*, *Instigations Methods*. Após a criação, é iniciada a verificação de cada *Rule* a fim de identificar seu estado inicial: aprovada ou reprovada.

Para cada sensor/atuador configurado com intervalo de leitura ou com leitura automática a cada mudança de valor, é instanciado um objeto da classe IoT.GPIO.Input.ISensorGPIO que efetua a leitura do valor do sensor, através da porta GPIO conectada, e preenche este valor no objeto *Sensor FBE*, disparando o processo de notificação das *Premises* e notificação entre ambientes, que por sua vez disparam todo o processo de inferência com a avaliação das *Rules* e consequente disparo das *Instigation* e *Methods*. O Código 8 apresenta o método que efetua a leitura do valor das portas GPIO do Raspberry.

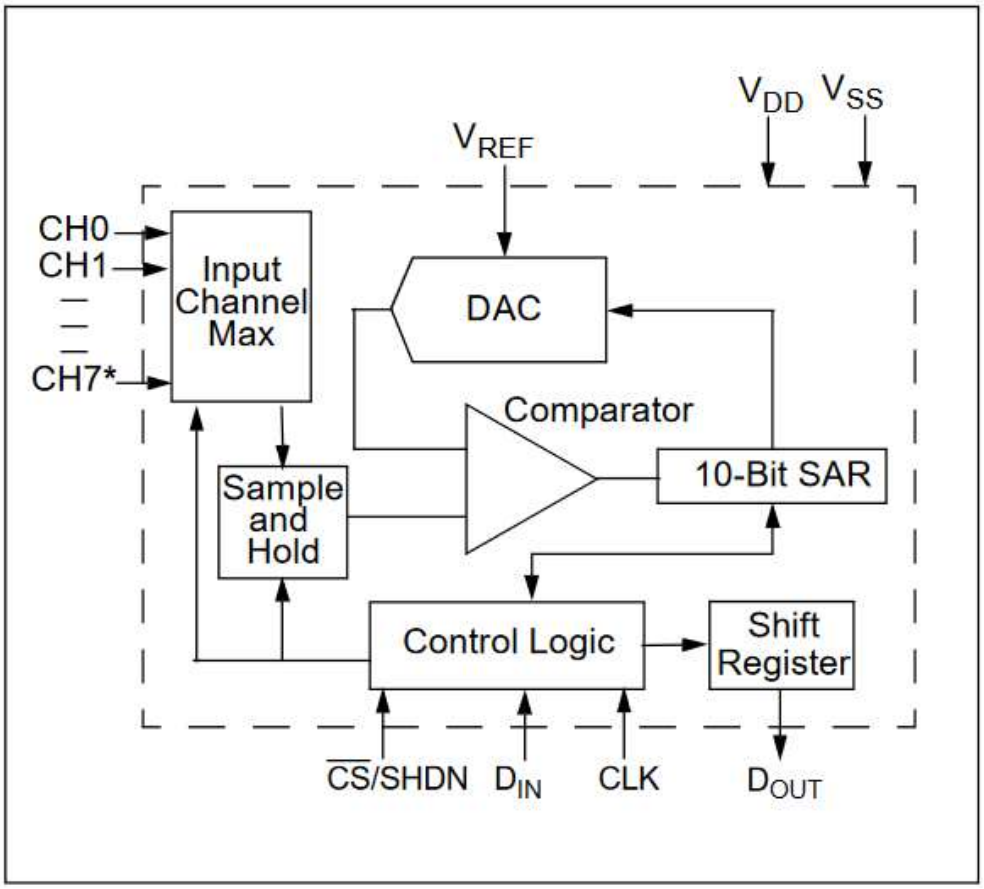
Código 8 – Trecho de código as classe GPIO, utilizado para efetuar a leitura das portas GPIO do Raspberry e enviar o valor para o SensorFBE do PON.IoT C#.

```
1  GpioController gpio = GpioController.GetDefault();
2  GpioPin pin = gpio.OpenPin(GPIOPort);
3  this.SensorController[0].SensorValue = pin.Read() == GpioPinValue.High ? 1 : 0;
4  if (PinMode.ToLower().StartsWith("output"))
5      pin.SetDriveMode(GpioPinDriveMode.Output);
6  else if (PinMode.ToLower().StartsWith("input"))
7      pin.SetDriveMode(GpioPinDriveMode.Input);
```

Fonte: Autoria própria.

O microcomputador Raspberry não possui portas analógicas; dessa forma, para conectar sensores que necessitam deste tipo de conexão, foi utilizado o conversor analógico/digital MCP3008 que possui 8 portas analógicas. Para configurar um sensor analógico no portal NOCS Server, basta definir o tipo do sensor (SensorType) como analógico e o número da porta definido no campo GPIO corresponderá ao número da porta analógica do MCP3008, podendo ser preenchido com valores de 1 a 8. A Figura 11 apresenta o diagrama funcional de blocos do conversor MCP3008, para fins de registro.

Figura 11 – Diagrama funcional de blocos do conversor analógico digital MCP3008



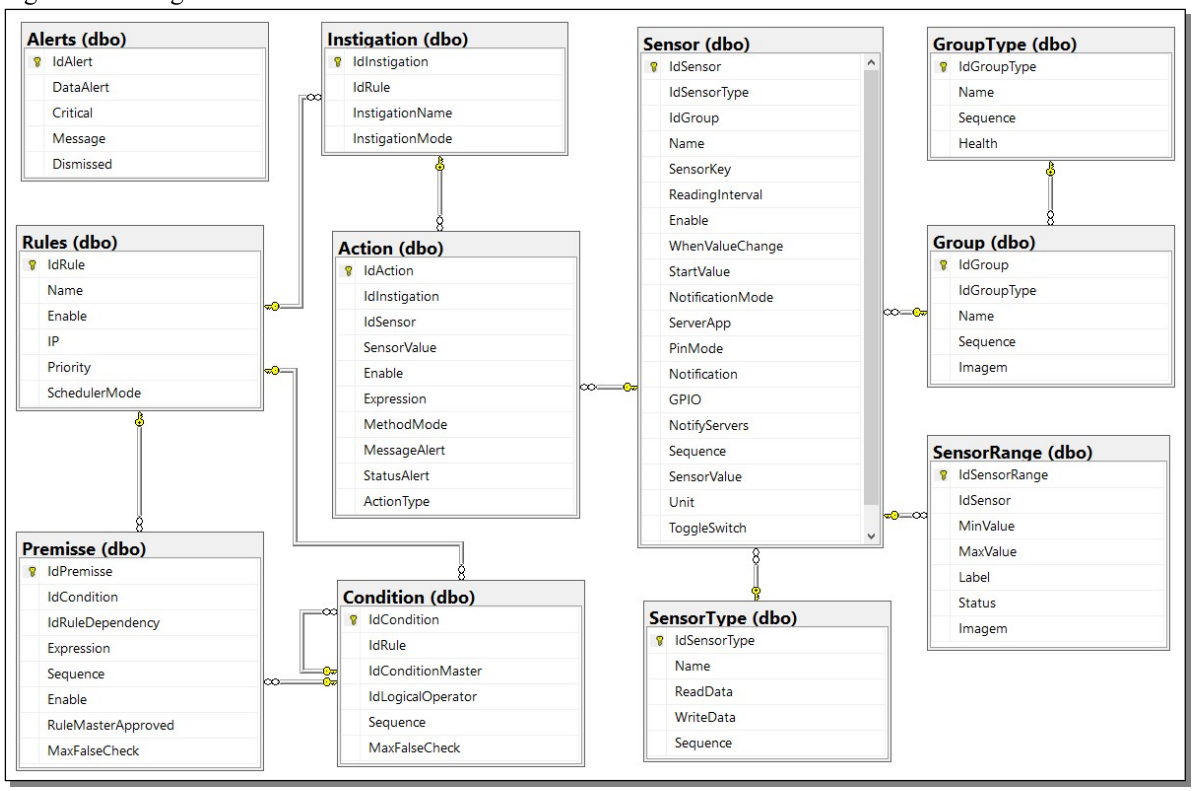
Fonte: Manual MCP3004/3008

4.3 PORTAL NOCS SERVER – NOTIFICATION ORIENTED CARE SYSTEM

Para a gestão de toda a solução, foi necessário criar um portal no qual centraliza-se toda a definição dos sensores e regras do ambiente, além de receber em tempo real todo o monitoramento de cada ambiente. O portal web denominado NOCS Server – *Notification Oriented Care System*, foi desenvolvido em ASP.NET com linguagem C# e permite a gestão e parametrização total dos ambientes controlados pelo sistema NOCS Control, construído sob o *Framework* PON.IoT C#, isso é possível pois no servidor do portal existe a persistência das configurações e dos dados de todos os ambientes.

Toda a estrutura do portal NOCS Server é publicada em serviço web na plataforma de nuvem da Microsoft, o Azure. Da mesma forma, a persistência dos dados é feita em banco de dados Azure Microsft SQL Server. A Figura 12 ilustra o diagrama de banco de dados das tabelas que compõem a persistência da aplicação NOCS. No diagrama, é possível visualizar as tabelas com suas respectivas colunas e o relacionamento entre elas.

Figura 12 – Diagrama de banco de dados das tabelas do NOCS Server



Fonte: Autoria própria

No diagrama da Figura 12, representa-se o agrupamento dos sensores (tabela *Sensors*) em grupos (*Group*), que estão classificados em tipo como Ambientes ou Indivíduos (*GroupType*). Os grupos possuem imagens, a fim de tornar a visualização pelo aplicativo NOCS Mobile mais amigável para o idoso, e possui um controle de sequência, permitindo priorizar a ordem de exibição de cada ambiente ou indivíduo no aplicativo. Os sensores/atuadores também possuem um relacionamento com os tipos de sensores (*SensorType*) para uma correta identificação do sistema no momento de criar o sensor físico vinculado ao Raspberry Pi. São exemplos de tipos de sensores: sensores digitais que fazem leitura e recebem apenas valores de 0 ou 1, sensores analógicos que permitem leitura dentro de um intervalo de valores, ou sensores específicos que utilizam mais de uma porta e envolvem cálculos matemáticos para identificação do valor de leitura, por exemplo o sensor HCSR04 (sensor ultrassônico de distância).

O sensor também possui sua lista de intervalos (*SensorRange*) que facilita a elaboração de *Rules* por pessoas que não são da área de tecnologia, pois cada faixa dos intervalos cadastrados possuem um rótulo de identificação, um *status* que identifica a criticidade daquela faixa (verde, amarelo, vermelho), e uma imagem para tornar amigável o monitoramento. Em um sensor que mede o nível de açúcar no sangue poder-se-ia ter os

seguintes intervalos, por exemplo: até 70 mg/dl um intervalo com status crítico (vermelho) e uma descrição de hipoglicemia, de 71 a 140 mg/dl um *status* normal (verde) com rótulo Normal e acima de 140 mg/dl outro *status* crítico com a descrição de hiperglicemia. Com esses intervalos, o gestor do ambiente pode facilmente criar uma *Rule* que envia um alerta caso o nível de açúcar do idoso esteja em hipoglicemia, por exemplo.

No diagrama do banco de dados, também é possível visualizar as tabelas que armazenam a estrutura das *Rules* com suas *Conditions*, *Premises*, *Instigation* e *Method*. Dentre os atributos da *Rule*, destaca-se o IP (ou nome de rede) do dispositivo Raspberry a qual ela está vinculada. Na *Condition*, tem-se um atributo recursivo, o IdConditionMaster que, quando preenchido, torna o registro uma *Subcondition* de outra *Condition*; dessa forma, é possível ter um aninhamento de *Conditions* de forma infinita.

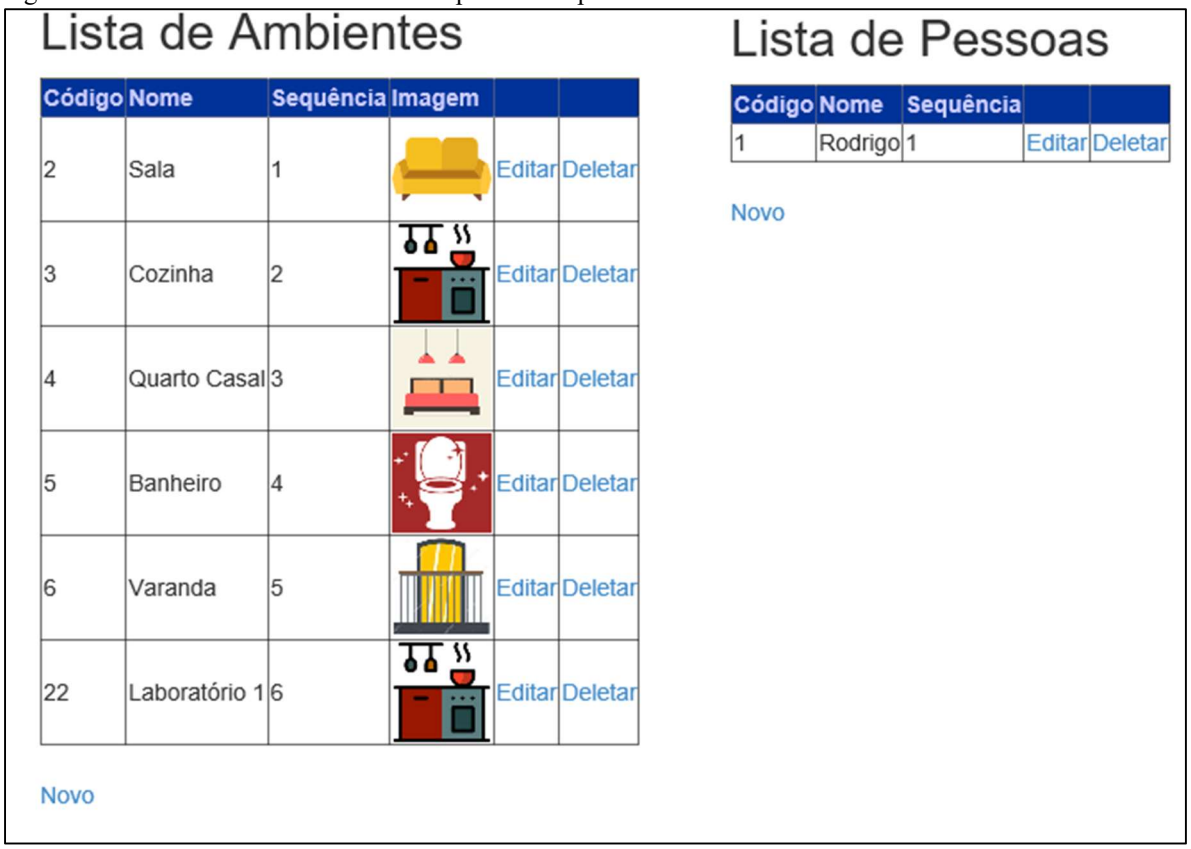
A tabela *Premise* possui um campo para controle do número máximo de avaliações com resultado falso (MaxFalseCheck), sendo que ao atingir esse máximo a *Premise* é desabilitada e outra *Premise* da mesma *Condition* é ativada; este mesmo campo existe na tabela de *Condition* e tem o mesmo papel. Este recurso é referente à Impertinência Dinâmica de *Premises* já apresentada na seção 4.1.5. A *Premise* também possui um campo para vínculo a uma *Rule* existente, permitindo, assim, *Premises* que dependem da aprovação de regras específicas.

A tabela de persistência das *Actions* possui um campo para definir o tipo da ação (*ActionType*), podendo ser a alteração em sensores/atuadores ou o envio de alertas. No caso de envio de alertas, essa tabela possui campos que definem a mensagem e o *status* do alerta que será enviado. Quando uma *Rule* é aprovada e uma das suas *Actions* é para envio de alerta, esse alerta fica registrado na tabela *Alerts* com informações de sua criticidade, data da geração do alerta, a mensagem do alerta e se a mensagem já foi dispensada pelo usuário.

Os registros de mudanças nos valores de sensores e atuadores, assim como as avaliações de cada *Premise*, *Condition*, verificação da aprovação da *Rule*, execução das *Instigations* e dos *Methods*, não estão armazenados no banco de dados de persistência das *Rules* e sensores. Contudo, esses registros são armazenados em arquivos de logs e que, em virtude do volume de dados gerado pelo monitoramento do ambiente, são importados em banco de dados com características de *Big Data*. Essas informações podem ser analisadas para geração de *insights* sobre o ambiente, identificação de padrões, treinamento de redes neurais e até mesmo para análise e evolução da própria plataforma.

O portal NOCS Server permite o gerenciamento de ambientes e pessoas, incluindo novos registros ou editando os registros existentes. Para os ambientes, é necessário informar a sequência do ambiente, para a sua priorização no aplicativo para celulares - o NOCS Mobile, informar o nome do ambiente e uma imagem para deixar a interface amigável e fácil de operar pelo idoso. Para o gerenciamento de pessoas, é necessário informar apenas o nome e a sequência de ordenação no NOCS Mobile. A Figura 13 apresenta, do lado esquerdo, a lista de ambientes cadastrados enquanto ao lado direito é apresentada a listagem de pessoas para edição.

Figura 13 – Gerenciamento de ambiente e pessoas no portal NOCS Server



Fonte: Autoria própria

No portal NOCS Server, é possível o cadastramento de todos os sensores com todas as informações necessárias para sua utilização no NOCS Control para controle do ambiente. Na Figura 14, apresenta-se a listagem de todos os sensores/atuadores cadastrados, assim como o grupo à qual ele está vinculado.

Figura 14 – Listagem dos sensores/atuadores cadastrados no portal NOCS Server

Lista de sensores/atuadores

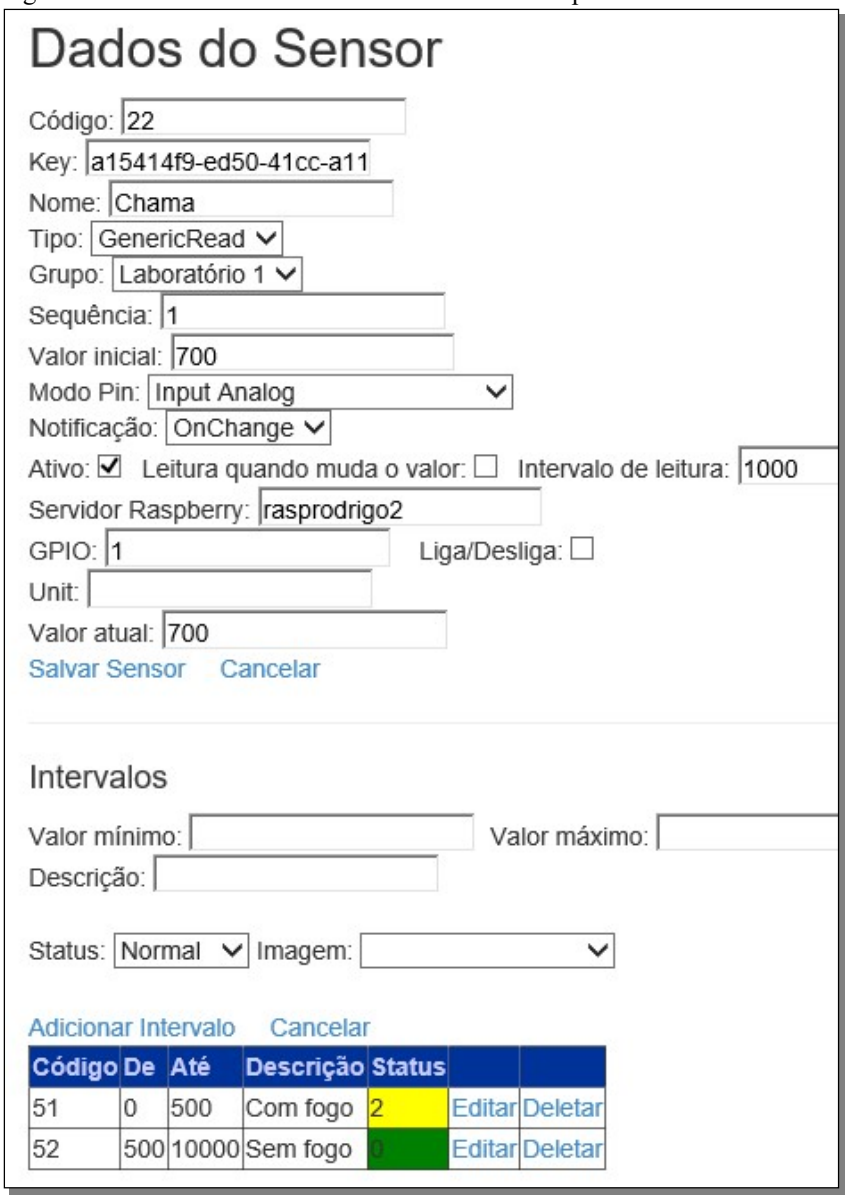
Novo

Código	Nome	Grupo		
1	Temperatuda do corpo	Rodrigo	Editar	Deletar
2	Glicose	Rodrigo	Editar	Deletar
3	Batimentos cardíacos	Rodrigo	Editar	Deletar
4	Lâmpada	Sala	Editar	Deletar
5	TV	Sala	Editar	Deletar
6	Porta	Sala	Editar	Deletar
7	Janela	Sala	Editar	Deletar
8	Lâmpada	Cozinha	Editar	Deletar
9	Fogão	Cozinha	Editar	Deletar
10	Porta	Cozinha	Editar	Deletar
11	Janela	Cozinha	Editar	Deletar
12	Lâmpada	Quarto Casal	Editar	Deletar
13	Ar condicionado	Quarto Casal	Editar	Deletar
14	Janela	Quarto Casal	Editar	Deletar
15	Abajur	Quarto Casal	Editar	Deletar
16	Lâmpada	Banheiro	Editar	Deletar
17	Chuveiro	Banheiro	Editar	Deletar
18	Torneira	Banheiro	Editar	Deletar
19	Lâmpada	Varanda	Editar	Deletar
20	Temperatura	Quarto Casal	Editar	Deletar
22	Chama	Laboratório 1	Editar	Deletar
23	Som	Laboratório 1	Editar	Deletar
24	LedVermelho	Laboratório 1	Editar	Deletar
25	LedVerde	Laboratório 1	Editar	Deletar
26	PIR	Laboratório 1	Editar	Deletar
28	Led1	Laboratório 1	Editar	Deletar
29	Led2	Laboratório 1	Editar	Deletar
30	Ultrasom	Laboratório 1	Editar	Deletar

Fonte: Autoria própria

Ao incluir um novo sensor ou atuador, o operador precisará definir todas as informações necessárias para a utilização deste dispositivo no NOCS Control, desde o nome do sensor, tipo, grupo que ele pertence, a sequência de exibição no NOCS Mobile, o intervalo de leitura, o nome ou IP do Raspberry que controlará o sensor/atuador físico, a porta GPIO que ele estará ligado no Raspberry, até mesmo a lista de intervalos de valores com seus respectivos rótulos, *status* e imagem. Na Figura 15, pode ser visualizada a interface de gerenciamento de sensores com dados de um sensor que identifica presença de chama em sua proximidade.

Figura 15 – Gerenciamento de sensores/atuadores no portal NOCS Server



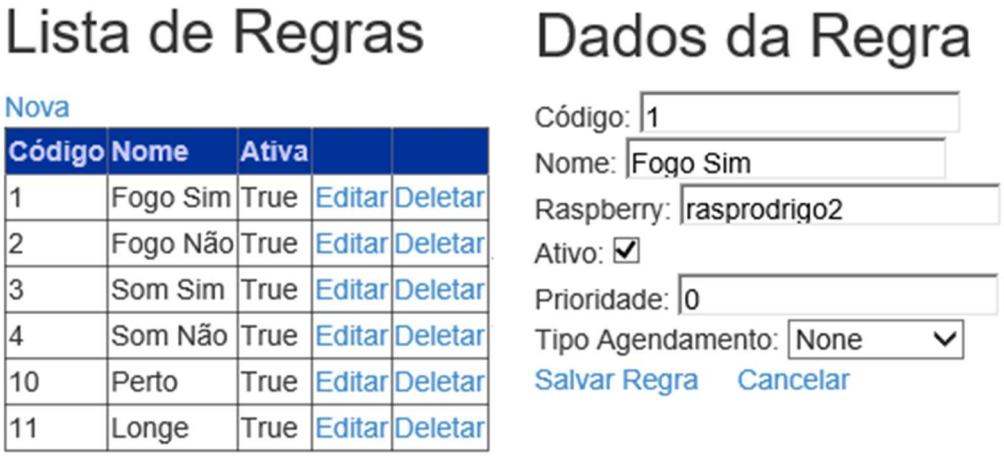
Fonte: Autoria própria

A principal funcionalidade do portal NOCS Server é o gerenciamento das *Rules* que atuam nos ambientes assistivos, modelando as *Conditions* e *Premises* para a aprovação da referida *Rule* e, para os casos de aprovação, as *Actions* que devem ser executadas no ambiente. Esse gerenciamento pode ser feito de forma dinâmica, por intermédio de uma interface que simplifica essa atividade.

Nos *frameworks* tradicionais do PON, a modelagem das *Rules* ocorre através de programação de código e uma posterior compilação que gera o código que controlará processo de inferência dessas regras (SIMÃO e STADZISZ, 2007). Também foi criado um *Wizard* que permite a configuração visual da *Rules* e geração automática do código fonte (VALENÇA, 2012). Contudo, nos sistemas de assistência à autonomia no domicílio,

conforme exposto, há necessidade de modelagem das regras em tempo de execução e de forma amigável para os usuários, pois a ideia é que os operadores que gerenciem as regras dos ambientes não necessitem ter conhecimento de programação. A Figura 16 apresenta a listagem das regras existente em uma residência, juntamente com os dados de uma dessas *regras*.

Figura 16 – Lista de regras e dados de uma regra no portal NOCS Server

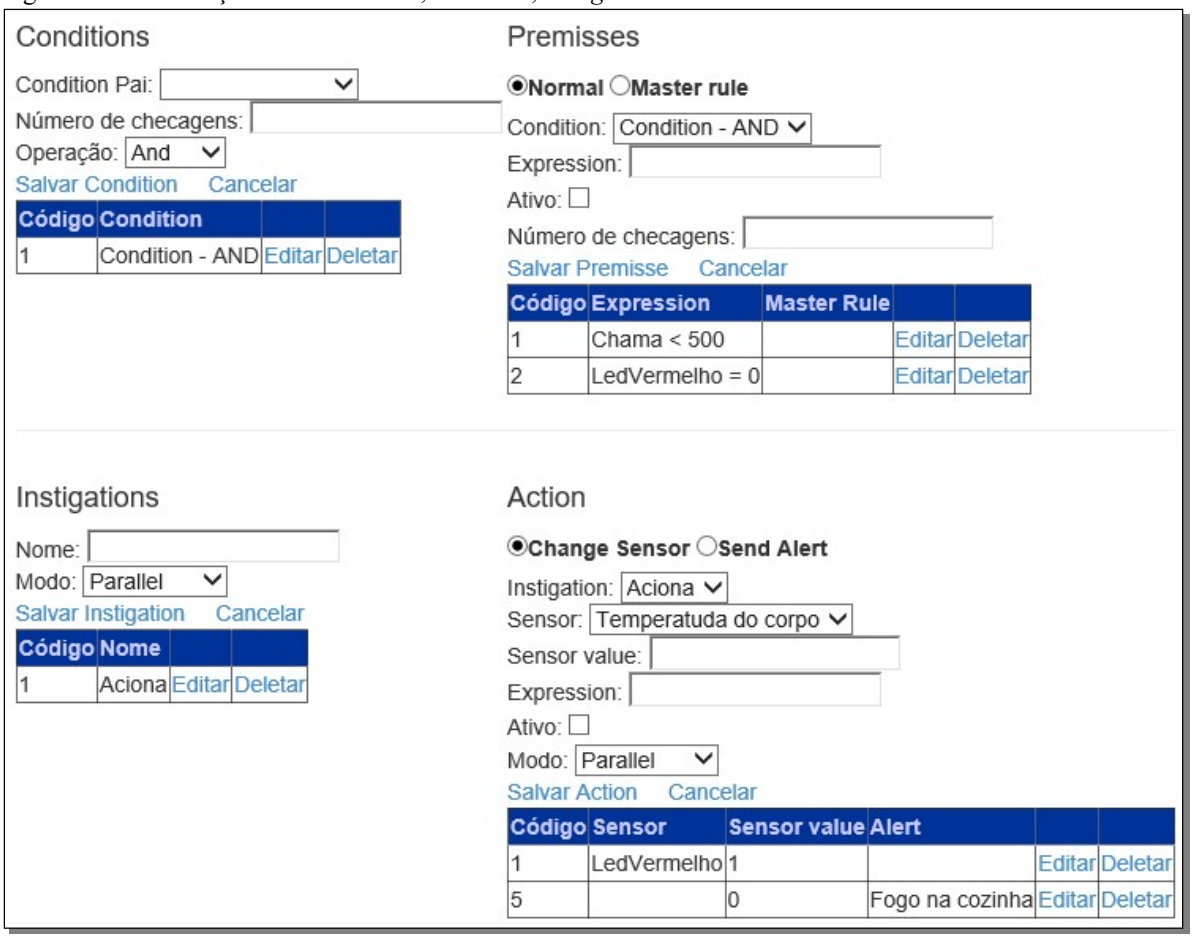


Fonte: Autoria própria

O preenchimento das partes que compõem uma *Rule* é feito visualmente de forma coesa, pois tem-se na interface de gerenciamento de *Rule*; logo após os dados da *Rule*, quatro grupos divididos em duas linhas com dois grupos cada. A primeira linha refere-se às condições para aprovação da *Rule*, composta pelos dados das *Conditions* e das *Premises*, enquanto a segunda linha refere-se às ações que devem ser executas na aprovação da *Rule*, com as informações das *Instigations* e dos *Methods*. A Figura 17 exibe os quatro grupos de dados, com seus respectivos campos.

Na Figura 17, exibe-se as informações da *Rule* que identifica presença de fogo no fogão da cozinha, e possui o papel apenas de alertar para o uso do fogão, além de manter uma lâmpada vermelha acessa para lembrar o idoso que algo está no fogão, e que precisa de constante monitoramento. O grupo da *Conditions*, para esta *Rule*, apresenta apenas uma *Condition* com operador lógico *And*, ou seja, todas as *Premises* vinculadas à essa *Condition* serão avaliadas através do operador *And*, tendo a *Rule* aprovada apenas nos casos que as duas *Premises* sejam verdadeiras. O grupo das *Premises* exibe duas expressões que devem ser avaliadas para aprovação da regra, sendo a primeira uma que compara a leitura do sensor de chama com valor menor que 500 (existência de chama), e outra que verifica se a lâmpada vermelha ainda não foi acionada, pois caso contrário, a *Rule* já teve sua ação executada.

Figura 17 – Informações da *Conditions*, *Premises*, *Instigations* e *Methods* da *Rule*



Fonte: Autoria própria

Também é possível visualizar na Figura 17 as informações referentes à execução dessa *Rule*, onde há apenas uma *Instigation* chamada "Aciona", e dois *Methods* que estão configurados para serem executados de forma paralela. O primeiro *Method* envia o valor 1 para a lâmpada vermelha (LedVermelho), e o segundo *Method* envia um alerta com status de atenção e com a mensagem "Fogão em uso".

A interface visual de gerenciamento das regras permite uma fácil visualização e entendimento do que cada regra monitora, suas condições para aprovação e as ações que serão executadas no ambiente.

O portal NOCS Server também disponibiliza via API REST métodos para que os microcomputadores que estejam executando o NOCS Control, busquem as definições de *Sensor FBEs* e *Rules*, as atualizações nos valores de *Sensor FBE*, as mensagens de alertas geradas, um método para dispensar o alerta e um método para notificar o portal NOCS Server sobre mudança no valor de algum sensor. O Código 9 apresenta o método utilizado para busca dos sensores e regras na inicialização de um microcomputador, esse método recebe

como parâmetro o IP do Raspberry e o tipo de busca, podendo ser I para a inicialização do ambiente, S para alterações em *Sensor FBEs* e R para alterações em *Rules*, e após a busca na estrutura de *Rules* para este ambiente, o retorno é serializado em formato JSON e enviado para o solicitante.

Código 9 – Trecho de código da classe wsController com o método que retorna JSON das definições de sensores e regras para um determinado Raspberry.

```
1  [HttpGet]
2  public string GetRaspDefinition(string IP, string Type)
3  {
4     SensorController controller = new SensorController();
5     if (Type == "I")
6        return JsonConvert.SerializeObject(controller.RaspDefinition(IP));
7     else
8        return        JsonConvert.SerializeObject(controller.RaspDefinitionChange(IP,
9  }
```

Fonte: Autoria própria.

4.4 NOCS MOBILE

Toda a estrutura do portal NOCS Server, assim como a aplicação NOCS Control permite o gerenciamento e o controle dos ambientes de forma simplificada e intuitiva. Contudo, não há muita praticidade em ter que acessar um portal apenas por meio de sua URL todas as vezes que o usuário do ambiente desejar alterar algum sensor/atuador ou verificar as condições do ambiente.

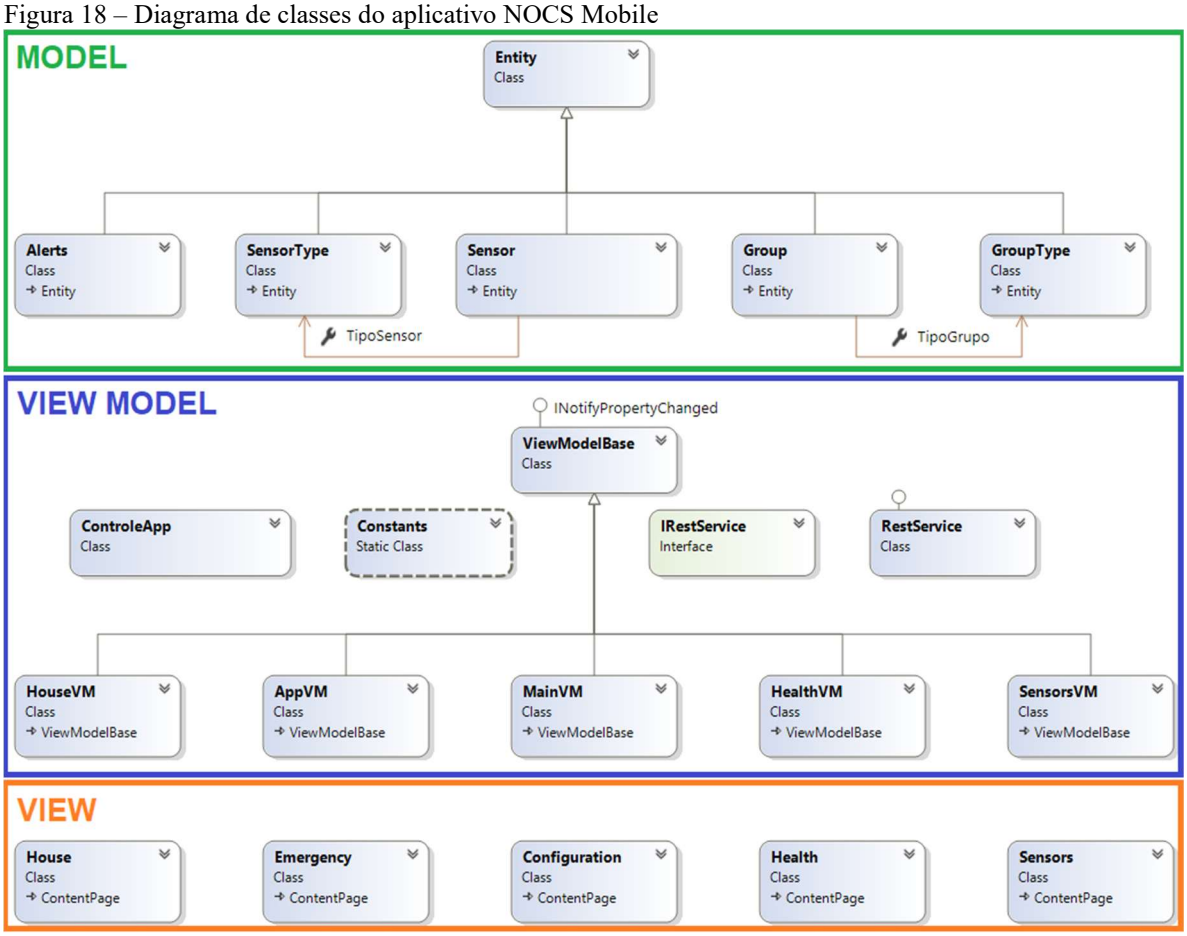
Com a finalidade de prover usabilidade e simplicidade para os usuários idosos do sistema NOCS, uma vez que a maioria deles possuem limitações físicas, cognitivas ou dificuldade de aprendizado de novas tecnologias, foi desenvolvida a aplicação NOCS Mobile para celulares e tabletes com sistema operacional Android ou IOS.

O NOCS Mobile teve seu desenvolvimento baseado no *framework* Xamarin da Microsoft que permite o desenvolvimento de aplicações para dispositivos móveis de forma nativa em linguagem C#, e que podem ser compiladas para dispositivos com sistema Android ou com sistema iOS (Iphones e Ipads). Sua interface foi construída pensando na usabilidade do aplicativo, utilizando botões com ícones em tamanho grande e com imagens intuitivas, facilitando a sua visão e identificação mesmo por pessoas com alguma limitação na visão.

A Figura 18 apresenta o diagrama de classes do aplicativo NOCS Mobile, onde é possível visualizar as classes que compõem a camada *Model* (entidades), as classes da

camada *ViewModel* (controle e validação das telas), e, por último, as classes referentes à camada *View* (telas). Portanto, o aplicativo NOCS Mobile é concebido utilizando o padrão de desenvolvimento MVVM (*Model-View- ViewModel).*

Figura 18 – Diagrama de classes do aplicativo NOCS Mobile



Fonte – Autoria própria

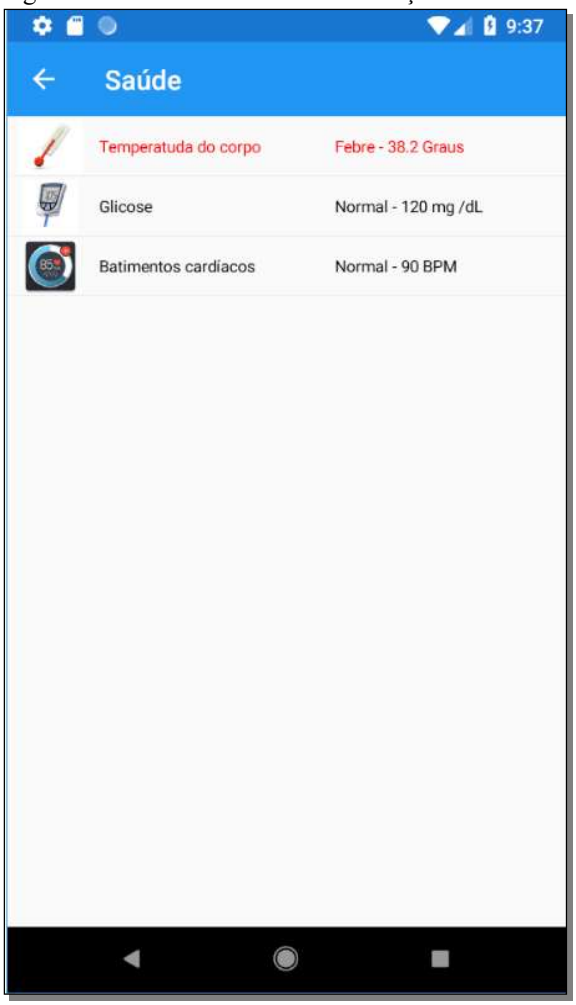
Ao abrir o aplicativo, a primeira tela exibe uma lista de notificações e alertas gerados pelas *Rules* do sistema NOCS. Como exemplo de notificações pode-se ter lembretes para o uso de algum medicamento ou consultas agendadas, enquanto os alertas podem ser referentes a condições adversas de saúde, como febres ou incontinências urinárias. A Figura 19 exibe a tela inicial do NOCS Mobile, onde além dos alertas e notificações também é apresentada a barra de ícones para as demais funcionalidades do aplicativo, como o monitoramento da saúde, o monitoramento da residência, um ícone de pânico para enviar uma mensagem aos familiares ou monitores em caso de quedas, por exemplo, e, por último, um ícone de configuração do aplicativo.

Figura 19 – Tela inicial do aplicativo NOCS Mobile sendo executa em dispositivo com sistema operacional Android

Fonte: Autoria própria

As duas principais funcionalidades do aplicativo são o monitoramento da condição de saúde do indivíduo e o monitoramento dos ambientes da residência. A tela de visualização das condições, chamada de Saúde, exibe a leitura atual de cada sensor que monitora sinais vitais, mostrando o nome do sensor, o valor da leitura, o rótulo dessa leitura e a criticidade da mesma (normal, atenção ou crítico). Na Figura 20, visualizam-se as condições de saúde referente à temperatura do corpo, o nível de açúcar do sangue e o total de batimentos cardíacos por minuto. Dentre as três leituras, tem-se uma situação crítica de Febre, referente à leitura da temperatura, e as outras duas leituras como normais.

Figura 20 – Monitoramento das condições de saúde do idoso no aplicativo NOCS Mobile



Fonte: Autoria própria

Ao clicar no ícone de monitoramento da residência, o aplicativo exibe a lista de todos os ambientes configurados, sendo cada um desses ambientes controlados por um Raspberry Pi rodando o NOCS Control. Os ícones dos ambientes são carregados dinamicamente, ou seja, podem ser trocados no portal NOCS, por meio do gerenciamento de ambientes, e essa substituição reflete imediatamente na tela do NOCS Mobile. Essa troca dos ícones tem a finalidade de deixar as imagens mais intuitivas para o idoso que utilizará o sistema, personalizando o sistema de forma a torná-lo o mais confortável possível para o usuário. A lista de ambientes da residência pode ser visualizada na Figura 21.

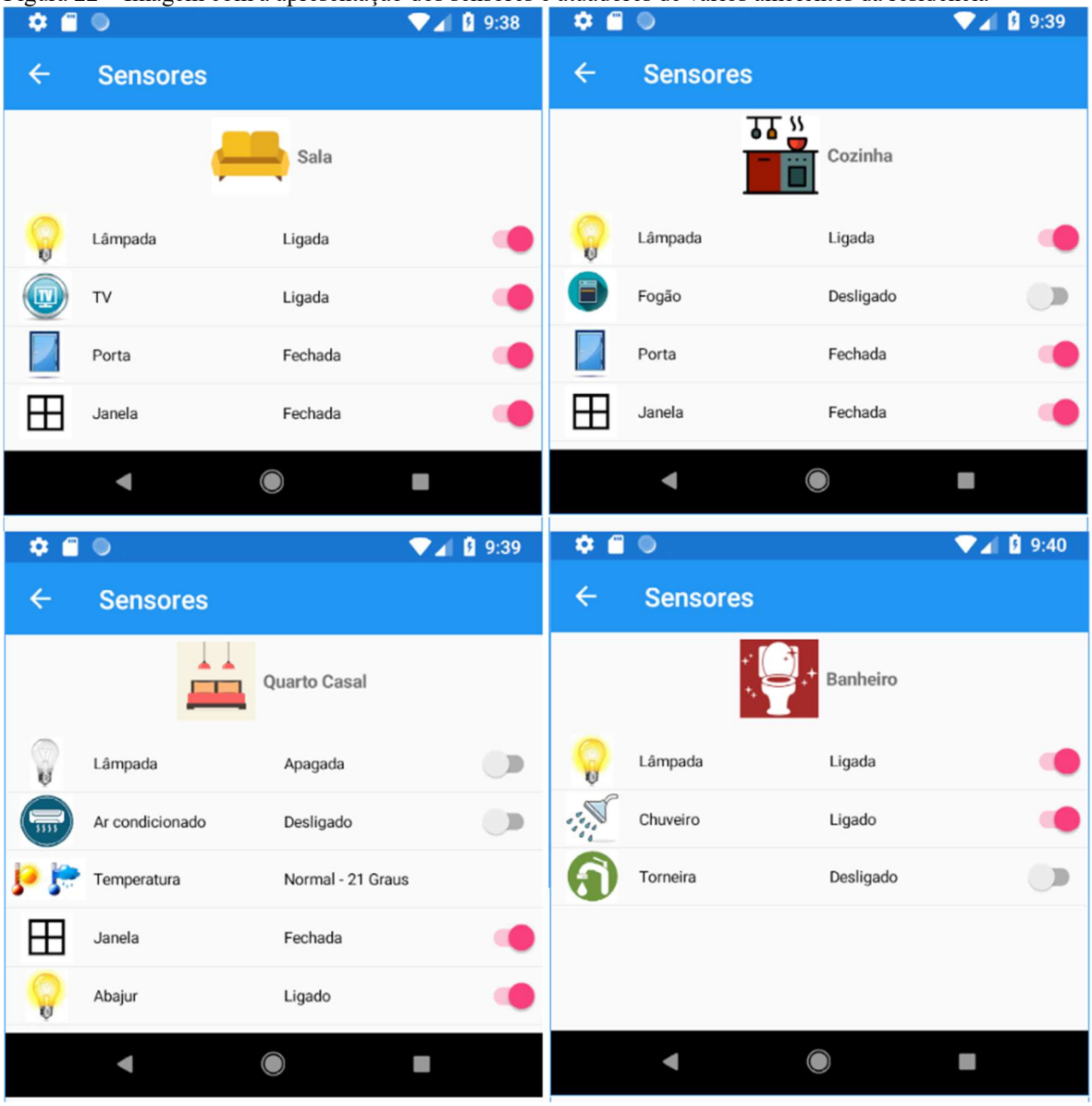
Figura 21 – Lista de ambientes da residência com seus respectivos ícones

Fonte: Autoria própria

Ao acessar cada ambiente da residência, é exibido todos os sensores e atuadores do respectivo ambiente, com o seu ícone, nome e situação, além de botão liga/desliga para os atuadores que permitem esse tipo de ação. Da mesma forma que o ícone do ambiente pode ser alterado no portal NOCS Server, os ícones dos sensores e atuadores também podem ser substituídos no portal pelo gerenciamento de sensores e intervalos, com a finalidade de tornar a interface mais amigável.

A Figura 22 mostra várias telas de ambientes distintos, onde para cada ambiente é exibido a lista de sensores e atuadores, com seus ícones, nome e situação atual. Para aqueles atuadores que permitem uma alteração de valores binários 0 ou 1 (liga/desliga ou aberto/fechado), a interface do ambiente exibe um botão para que o usuário possa efetuar a ação de troca do valor do atuador.

Figura 22 – Imagem com a apresentação dos sensores e atuadores de vários ambientes da residência



Fonte: Autoria própria

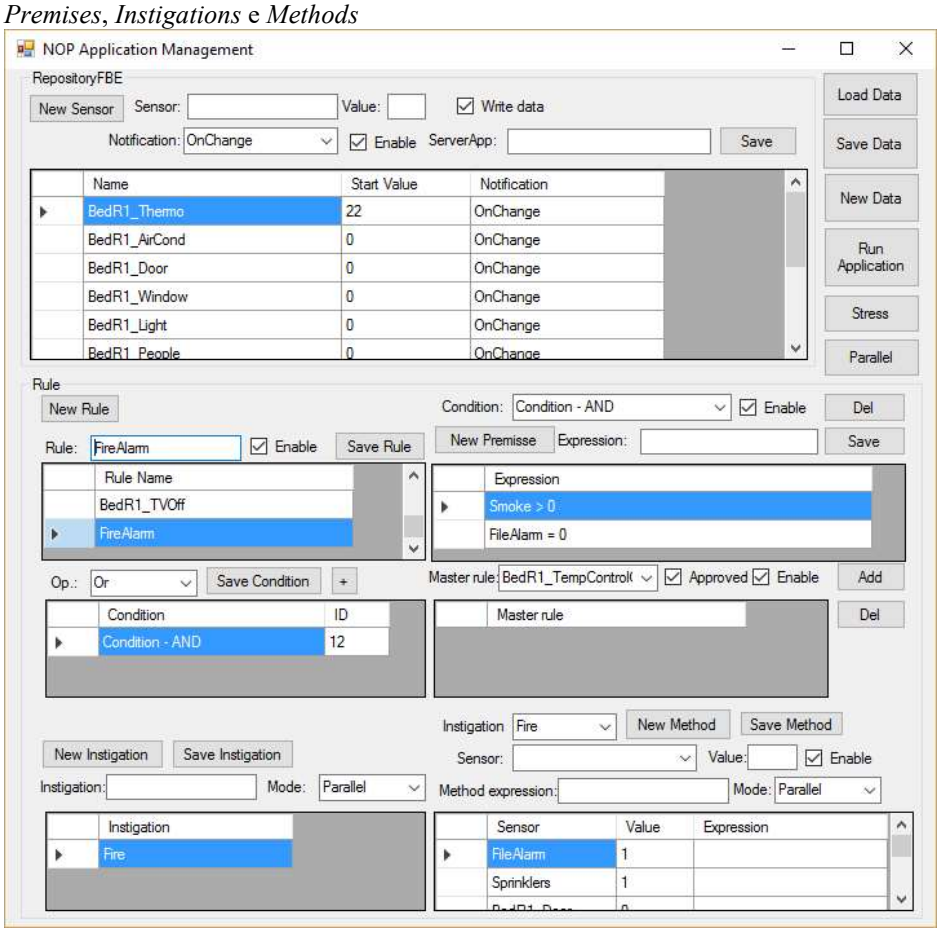
No momento em que o operador efetua uma ação na interface de sensores, como, por exemplo, o acender de uma lâmpada, o aplicativo envia o comando de mudança de valor para o portal NOCS Server que, ao receber persiste essa mudança em log interno, envia a notificação de rede para o microcomputador responsável por esse sensor/atuador. Assim que o Raspberry Pi recebe a notificação, ele atualiza o novo valor no objeto da classe *Sensor FBE* do respectivo sensor, que, consequentemente, envia esse valor para o dispositivo ligado ao Raspberry Pi através da porta GPIO e dispara as notificações de *Premises*, com a finalidade de verificar aprovações de *Rules* dependentes da mudança desse sensor, iniciando, assim, um ciclo de avaliação e possíveis mudanças no ambiente por intermédio dos demais sensores e atuadores.

4.5 SIMULADOR DE AMBIENTES

Com a dificuldade de instalação do sistema em uma residência real com um morador idoso, foi necessária a criação de uma interface para simular e forçar alteração nos valores dos sensores. O primeiro passo para simular um cenário é executar a importação das definições do ambiente a ser testado. Essa importação pode ser feita por meio da API REST disponibilizada pelo portal NOCS Server, ou por meio de arquivos de dados que podem ser carregados e salvos pelo simulador. Ao importar as definições, o sistema de simulação exibe as informações dos sensores e atuadores do ambiente, as *Rules* com suas *Conditions*, *Premises*, *Instigations* e *Methods*, permitindo alteração desses parâmetros, inclusive com a simulação em execução, validando também o requisito de reconfiguração dinâmica de ambientes.

A Figura 23 apresenta a interface do gerenciamento visual dos *Sensor FBEs* e as *Rules* com seus respectivos elementos. Nessa interface, é possível carregar as configurações previamente salvas em arquivo, alterar qualquer parâmetro e iniciar o monitoramento do ambiente. Na imagem está selecionada uma *Rule* de alarme de incêndio que verifica a presença de fumaça e se ainda não foi disparado o alarme de incêndio. Caso estas *Premises* sejam verdadeiras, a *Rule* será aprovada e acionará, através da *Instigation* “Fire”, o alarme de incêndio, os chuveiros (*sprinkers*) e abrirá as portas e janelas.

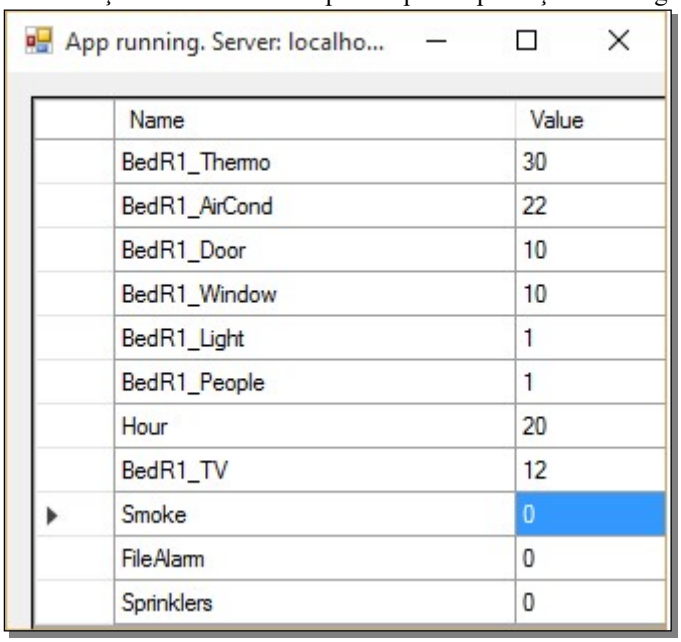
Figura 23 – Interface visual do gerenciamento dinâmico de sensores/atuadores (FBE), *Rules*, *Conditions*, *Premises*, *Instigations* e *Methods*



Fonte: Autoria própria

Para a alteração dos valores de todos sensores e atuadores do ambiente, foi desenvolvida uma interface visual de simulação da execução do ambiente, apresentada na Figura 24, na qual é possível alterar valores nos sensores e analisar as ações tomadas no ambiente, inclusive nas situações onde o sensor de um ambiente pode afetar o estado de outro ambiente, sendo possível testar o requisito de distribuição nos sistemas sencientes desenvolvidos com o *Framework* PON.IoT C#. Durante a execução da simulação do ambiente é possível alterar as *Rules*, *Conditions*, *Premises*, *Instigations* e *Methods*, por intermédio da interface de gerenciamento, e acompanhar estas alterações na simulação, em tempo real, sem a necessidade de qualquer parada no sistema.

Figura 24 – Interface do simulador de ambientes para alteração dos valores dos sensores e acompanhamento das alterações no ambiente disparada pelas aprovações nas regras



Fonte: Autoria própria

Todas as alterações que ocorrem na simulação do ambiente são registradas em arquivos de log, seja ela alteração no valor de sensores, avaliações de *Premises*, *Conditions*, aprovações ou reprovações das *Rules*, execuções de *Instigations* e as execuções dos *Methods*. Esses logs podem ser importados em banco de dados não relacional para análises do resultado da simulação. A Tabela 7 apresenta alguns exemplos das linhas registradas nos arquivos de logs. Esses registros utilizam um símbolo para indicar o gerador do log, como, por exemplo, o símbolo + (mais) para registros gerado a partir da avaliação de *Premises*.

Tabela 7 – Exemplos de registros de logs da simulação do ambiente.

Registros de log
+ Premise: 2 Expression: Smoke > 0 Result: True (Was False) - Sensor Value: 12
* Condition: 10 Rule: 6 Logical Operator: AND Result: True (Was False) Elements: 3 Elements true: 2
Rule: 6 Fire Approved (Was False) Executing: True
$ Method: 9 Target: FireAlarm Value: 1 (Was 0)

Fonte: Autoria própria.

Vários cenários foram simulados para a elaboração desse trabalho, inclusive os ambientes apresentados no NOCS Mobile tiveram seu funcionamento avaliado neste simulador. Outros cenários foram elaborados para validação de sensores e *Rules* específicas como, por exemplo, um cenário onde os seguintes sensores/atuadores foram utilizados: termômetro, ar-condicionado, porta, janela, luz, presença de pessoa no ambiente, hora do dia,

controle da TV, sensor de fumaça, alarme de incêndio e chuveiros antichamas (*sprinklers*). Com estes sensores/atuadores foram criadas *Rules* que controlam a temperatura do ambiente, a iluminação, a televisão e situações de incêndio, um exemplo das *Premises* de uma destas *Rules* é o controle da iluminação do ambiente que acende as luzes se as mesmas estiverem apagadas, se houver alguém no ambiente e se o horário for entre 18 e 23 h. Com essa configuração, pode-se forçar a mudança do horário do dia para verificar o acionamento da lâmpada sem a necessidade de aguardar um horário específico.

5 RESULTADOS

O sistema desenvolvido para assistência à autonomia no domicílio com a utilização do Paradigma Orientado a Notificações permitiu a inclusão de sensores, criação e alteração de *Rules*, mudança nos operadores lógicos das *Conditions*, alteração nas expressões das *Premises*, criação e alteração nas *Instigations* e *Methods*, inclusive na forma de execução (sequencial ou paralela), e alteração nos valores enviados aos atuadores através dos *Methods*. Todas estas alterações foram executadas com a aplicação rodando, podendo ser executadas facilmente por um enfermeiro ou o gestor do ambiente inteligente.

A materialização do PON, por meio do *Framework* PON.IoT C#, permitiu o gerenciamento dos sensores e atuadores e a execução de todo processo de inferência das *Rules* nos microcomputadores Raspberry Pi, além de estabelecer a comunicação entre ambientes por meio das notificações de rede, distribuir as execuções entre os núcleos do processador e permitir uma modelagem do negócio como regras. O *Framework* PON.IoT C# suporta uma variedade de sensores e atuadores, inclusive sensores analógicos que podem ser ligados ao conversor analógico/digital MCP3008.

O portal NOCS Server possibilitou autonomia e facilidade no gerenciamento da residência, com seus respectivos sensores, ambientes, pessoas e toda a estrutura das *Rules*, além de armazenar toda a persistências das configurações do sistema e disponibilizar essa informação por meio de uma API REST. A Figura 25 apresenta o resultado do teste de *stress* feito com a utilização da ferramenta Apache JMeter, no qual foram testados 121 requisições de configuração do ambiente, com um total de 10 ambientes simultâneos. O tempo médio de retorno da solicitação foi de 5719 ms, proporcionando um total de aproximadamente 30,9 requisições por minuto.

Figura 25 – Resulta do teste de *stress* para a busca das definições na API REST do ambiente no servidor NOCS Server

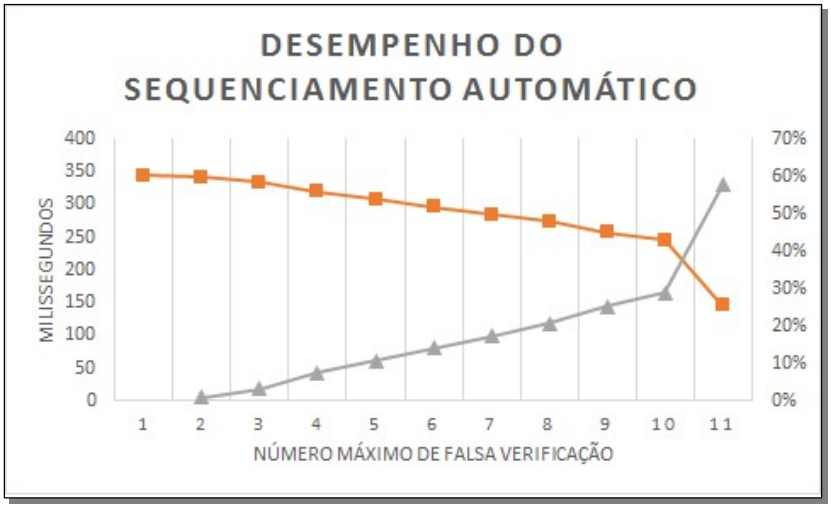
# Samples	Average	Min	Max	Error %	Throughput	Received KB/sec	Sent KB/sec
121	5719	4594	10903	0.00%	30.9/min	5.60	0.08

Fonte: Autoria própria

A Figura 26 apresenta o resultado do teste de desempenho do sequenciamento automático de *Premises* e *Subconditions*, na qual se tem no eixo horizontal as configurações utilizadas no sequenciamento automático, sendo a primeira com este recurso desabilitado e os demais com a definição do número máximo de notificações desnecessárias variando de dez a uma. No eixo vertical principal (eixo da esquerda) tem-se a linha laranja que apresenta

o tempo total de processamento do cenário proposto no teste em milissegundos, enquanto no eixo vertical secundário (eixo da direita), apresentado com a linha cinza, temos o percentual de ganho de desempenho da aplicação em cada cenário, comparado ao cenário onde esta funcionalidade está desabilitada.

Figura 26 – Resultado do teste de sequenciamento automático de *Premises* onde a linha laranja representa o tempo de execução em relação ao número máximo consecutivo de notificações sem alteração da *Premise* enquanto a linha cinza apresenta o ganho de desempenho em cada situação comparando com a situação onde este recurso não é utilizado



Fonte: Autoria própria

Na Tabela 8, apresenta-se o resultado do teste de *stress* de monitoramento de ambientes pelo microcomputador Raspberry. Nesse teste, coletou-se o número de leituras de sensores, avaliações de *Premises* e o tempo total de cada conjunto de ambientes. De acordo com os dados apresentados, é possível calcular a média de 235 notificações de sensores por segundo, e a média de 415 avaliações de *Premises* por segundo, totalizando uma média de 650 execuções por segundo.

Tabela 8 – Resultado do teste de *Stress*.

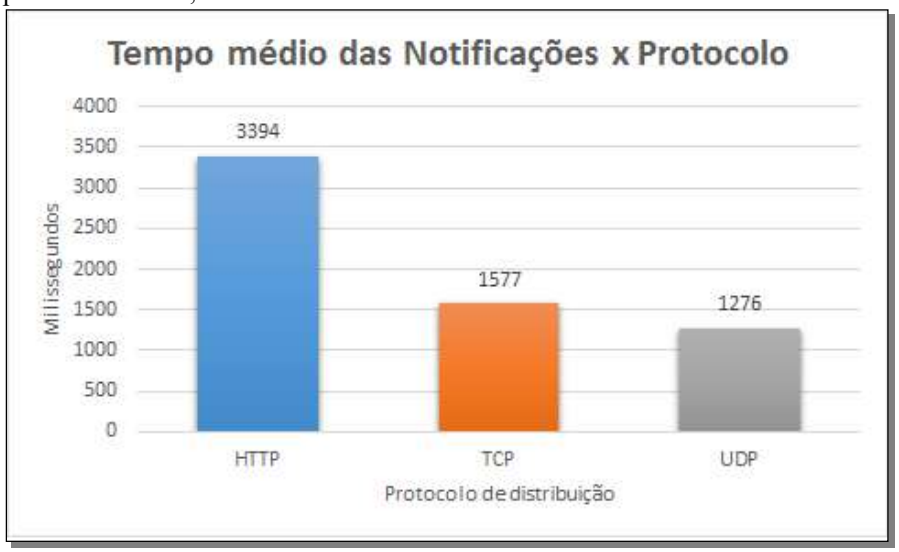
Ambientes	***Leituras***	***Premises***	***Tempo (s)***
1	341	600	1,47
5	1.705	3.000	7,15
25	8.525	15.000	35,82
50	17.050	30.000	72,87
100	34.100	60.000	144,13

Fonte: Autoria própria.

Para este trabalho, optou-se o protocolo TCP para as notificações entre os ambientes. Contudo, foram executados testes também com os protocolos UDP e HTTP, sendo que o protocolo UDP ficou com o menor tempo de envio, mas sem uma garantia de entrega, enquanto o protocolo HTTP foi o que apresentou o maior tempo de notificação.

A Figura 27 apresenta os resultados do teste de envio, com os protocolos HTTP, TCP e UDP, sendo processado envio de cem notificações entre ambientes diferentes, nos quais os microcomputadores Raspberry Pi estavam conectados em uma mesma rede.

Figura 27 - Resultado do teste de desempenho no envio de cem notificações entre ambientes, utilizando os protocolos HTTP, TCP e UDP



Fonte: Autoria própria

No próximo capítulo, discutem-se os resultados apresentados no capítulo atual, verificando se os resultados obtidos são aderentes aos requisitos de sistemas AAL.

6 DISCUSSÃO

O sistema AAL proposto nessa dissertação permitiu o gerenciamento do ambiente, por meio do *Framework* PON.IoT C#, com total controle dos cômodos, desde sua configuração, persistência, monitoramento e atuação. Além de possibilitar a execução do PON nos microcomputadores Raspberry Pi, permitindo uma automatização no controle de sensores e atuadores, e uma abstração na utilização dos recursos de paralelismo e distribuição.

O trabalho de Rashidi e Mihailidis (RASHIDI e MIHAILIDIS, 2013) apresenta soluções AAL aplicáveis a cenários específicos de assistência à autonomia como, por exemplo, monitoramento da qualidade de sono, detecção de quedas ou assistência à idosos com demência. Todavia, este trabalho propõe uma solução generalista que, apesar de basear-se em regras específicas para cada necessidade, compartilham vários cenários e atuações em uma mesma solução, facilitando, assim, o gerenciamento e monitoramento de todo o ambiente e atuando como um agrupador de tecnologias e sistemas específicos de assistência domiciliar.

No sistema desenvolvido nesta dissertação, foi possível a aplicação de testes do sistema de uma forma geral, sem a necessidade de instalação física em uma residência, por meio do simulador de ambientes que permite uma configuração rápida de cenários, e o monitoramento da execução das regras com a finalidade de garantir a correta leitura e ações no respectivo ambiente.

O desempenho do portal NOCS Server, avaliado por meio do teste de consultas das definições dos ambientes, efetuada pelos microcomputadores Raspberry Pi, obteve resultado de aproximadamente 30,9 requisições por minuto. Com esse resultado, pode-se inferir que uma residência com 15 ambientes levaria em torno de 30 s para estar com todo o monitoramento funcionando completamente.

O resultado do teste proposto de sequenciamento automático apresenta um ganho gradativo no desempenho, sendo que o tempo de processamento pode chegar a uma redução de 60% na comparação da execução da *Rule* sem o uso desta funcionalidade com a execução que efetua o re-sequenciamento após a primeira notificação desnecessária, ou seja, que não altera o valor da *Premise*. Neste último caso, o sistema efetua apenas uma notificação à *Premise*, desabilitando a mesma, sendo que as próximas alterações no valor do seu sensor

não disparam nenhum processo de notificação, melhorando diretamente a utilização dos recursos do dispositivo.

No teste de *stress* das notificações entre ambientes distintos, foi possível processar centenas de notificações por segundo e levando-se em consideração que cada ambiente será controlado por um microcomputador, o tempo de resposta obtido está coerente com ambientes reais, onde os intervalos de notificação da maioria dos sensores ocorrem em intervalos maiores, garantindo, assim, a execução das regras em menos de um segundo, fator este fundamental principalmente nas questões relacionadas à segurança, como incêndios, quedas do idoso, ou problemas de saúde como uma parada respiratória.

No teste de notificações, o tempo de notificação do protocolo HTTP foi, em média, o dobro do tempo utilizado pelos protocolos TCP e UDP, um resultado esperado em virtude das particularidades de comunicação de cada protocolo. Contudo, a capacidade média de notificações nos protocolos HTTP, TCP e UDP foram, respectivamente, de 30, 63 e 78 notificações por segundo, sendo satisfatórios para envio de comando aos sensores que estão em outros ambientes. Ao se comparar este resultado com o trabalho de Kruger e Hancke (KRUGER e HANCKE, 2014), constatou-se desempenho dez vezes melhor na comunicação de rede, ocasionado, principalmente, pela versão mais atual do microcomputador, pelo sistema operacional utilizado e pela redução no tamanho do pacote de notificação utilizado no *Framework* PON.IoT C#.

O aplicativo para dispositivos móveis NOCS Mobile permite maior usabilidade para os usuários do sistema, principalmente aqueles com algum tipo de limitação. Sua interface foi concebida com conceitos de UX – *User Experience* (Experiência do Usuário), com a utilização de ícones intuitivos, botões grandes para telas sensíveis ao toque. Segundo o trabalho de Rashidi e Mihailidis (RASHIDI e MIHAILIDIS, 2013), além da interface com o usuário ser simplificada em virtude de suas limitações, a solução AAL não deve gerar desconforto ou limitação de movimentos, em virtude dos sensores vestíveis responsáveis pelo monitoramento das condições de saúde, do posicionamento e movimentação do idoso. A capacidade de inclusão dinâmica de novos sensores, proposta neste trabalho, possibilita uma rápida reconfiguração do sistema para utilização de sensores menores e mais modernos, minimizando assim, estes desconfortos ocasionados pelo monitoramento do idoso.

Os requisitos da computação senciente, como distribuição e paralelismo, foram comprovados no ambiente simulado por meio da distribuição das tarefas (notificações) entre todos os núcleos de processadores do equipamento, e nas situações em que a leitura de um

sensor alterava o estado de outro ambiente por meio de notificações pela rede entre os microcomputadores. A capacidade de o sistema atuar de forma distribuída permite a utilização da tecnologia no monitoramento de ambientes compartilhados como lares, asilos, condomínios, bairros ou até mesmo cidades inteiras.

O uso do PON em sistemas sencientes foi sugerido por Simão *et al*. (SIMÃO, RENAUX, LINHARES e STADZISZ, 2014), e este trabalho é a primeira implementação real do uso deste paradigma em ambientes inteligentes, gerando, assim, resultados inéditos para a literatura. Os resultados obtidos no processamento das *Rules*, leitura dos sensores e notificações entre ambientes, com execuções em milissegundos, são adequados quando comparados às demandas dos cenários reais dos ambientes inteligentes, e indicam eficiência em aplicações para sistemas inteligentes desenvolvidos sob o PON.

A implementação do *framework* PON.IoT C# possibilita que novas pesquisas possam utilizar o PON em sistemas para ambientes inteligentes, pois além de abstrair o uso de recursos como paralelismo e distribuição, permite que sistemas criados sob este *framework* possam ser executados em microcomputadores como o Raspberry Pi. Neste trabalho, também se propôs a modelagem das *Rules* em tempo de execução, que é inédita nas pesquisas realizadas com PON. Esse recurso poderá ser explorado nas mais variadas aplicações do PON.

A utilização de sistemas AAL que auxiliem o idoso na execução das principais atividades diárias, monitorando suas condições de saúde e as condições do seu ambiente, é fundamental para o aumento na qualidade de vida e nas condições de saúde da população idosa em diferentes níveis econômicos e sociais.

Para a disseminação da tecnologia, é importante que o sistema se adeque às mais variadas situações, perfis e condições, tanto do ambiente como das pessoas nele inseridas. Esta adequação deve ser intuitiva, transparente e rápida, além de permitir a incorporação de novas tecnologias de sensoriamento dos ambientes inteligentes.

A criação de uma aplicação que gerencie, armazene, monitore e simule ambientes inteligentes para assistência à autonomia no domicílio, com a utilização do Paradigma Orientado a Notificações, permite o auxílio na execução das tarefas cotidianas dos idosos e demais pessoas com necessidade de autonomia. Neste cenário, a utilização do PON no desenvolvimento deste sistema contribuiu para uma implementação robusta e dinâmica, possibilitando, de uma forma natural, a criação e as adequações das *Rules* conforme o

contexto e a situação, a execução de tarefas paralelas e a distribuição do sistema de forma lógica.

Os recursos avaliados no desenvolvimento do sistema NOCS, tais como extensibilidade, controle de dispositivos com microcomputadores (IoT) e monitoramento da condição de saúde, são diferenciais que permitem a criação de um sistema que vai de encontro às necessidades da população de idosos que padece de auxílio em suas atividades diárias e aguardam que as soluções desta natureza, sejam realidades em suas vidas.

7 CONCLUSÕES

Para atender ao objetivo específico de desenvolver um sistema de sensoriamento múltiplo que permite a inclusão e a alteração de sensores e regras em tempo de execução de forma simplificada, a aplicação servidor, o NOCS Server, possibilitou o gerenciamento centralizado e intuitivo para inclusão de novos sensores/atuadores, ou alteração nas "regras do ambiente". Em virtude do alto nível na modelagem das regras, o gerenciamento pode ser feito por usuários sem conhecimento técnico, como profissionais de saúde, familiares do idoso, ou até mesmo o próprio residente.

A abstração da comunicação entre os sensores físicos e o microcomputador Raspberry, onde se torna necessário apenas configurar parâmetros básicos como tipo de sensor, intervalo de leitura e porta ligada ao sensor, permite que a capacidade do sistema AAL NOCS seja extensível a sensores não testados, ou ainda, não desenvolvidos, pois o *Framework* PON.IoT C# estabelece a comunicação, seja ela analógica ou digital, de forma transparente com estes sensores.

Em relação ao segundo objetivo específico, de desenvolver uma aplicação para dispositivos móveis para a interação com o ambiente, o NOCS Mobile ofereceu gerenciamento total da residência, monitoramento das condições de saúde e exibição de alertas e notificações, isso tudo sendo executado no dispositivo mais utilizado no dia a dia: o celular, podendo inclusive ser adaptado para execução em relógios inteligentes (*smartwatches*).

O terceiro objetivo específico, relacionado ao desenvolvimento de um simulador de ambientes, necessário para a avalição, testes e validação de toda a solução proposta, foi atendido por meio do simulador de sensores e ambientes, que permitiu simulações rápidas com a possibilidade de reconfiguração das *Rules* e de seus elementos e o acompanhamento dos logs de execução, possibilitando uma análise das situações nas quais *Rules* específicas não foram acionadas conforme o previsto.

Além da nova materialização do PON, por meio do *Framework* PON.IoT C#, outras contribuições para este paradigma foram obtidas, como a implementação de impertinência dinâmica para *Premises* e *Subconditions*; o re-sequenciamento automático de *Premises* e *Conditions*; a capacidade de aninhamento infinito de *Subconditions*; a capacidade de alteração das *Rules* em tempo de execução; a validação da capacidade do PON em ser paralelo e distribuído em aplicações reais; e a própria utilização do PON na elaboração de

um sistema voltado para auxilio a pessoas idosas. Essas contribuições atendem ao último objetivo específico dessa dissertação, que consiste em avaliar o uso do PON em sistemas AAL.

Em relação ao objetivo principal dessa dissertação, conclui-se que com a utilização do PON foi possível criar uma solução AAL extensível, pois ela não está vinculada a um conjunto de sensores específicos ou regras pré-determinadas. Pelo contrário, o PON permite o vínculo de novos sensores ao sistema. As “regras de ambiente” também podem ser reconfiguradas em tempo real, pois por meio das *Rules* do PON, o sistema AAL habilita a atualização da estrutura dessas *Rules* permitindo reconfigurações totais do ambiente em segundos. Além disso, o sistema AAL, resultado desta pesquisa, monitora o ambiente com o uso constante do paralelismo e do processamento distribuído, proporcionando um melhor desempenho da solução.

8 TRABALHOS FUTUROS

O desenvolvimento de sistemas AAL, assim como a aplicação do PON na computação senciente, são temas extremamente abrangentes. Neste âmbito, a solução proposta nessa dissertação possui vários pontos que devem ser implementados e melhorados. Nesse capítulo, apresentam-se as seguintes sugestões de possíveis trabalhos futuros relacionados a essa pesquisa:

1. apesar de toda dificuldade de testes em ambientes reais utilizando a solução NOCS, tendo em vista os custos envolvidos na automação de uma residência e na resistência dos proprietários idosos que se sentem "invadidos" pelo monitoramento constante, para a própria evolução desse trabalho será necessária sua aplicação em uma residência. O resultado desse ensaio trará dados que poderão ser analisados a fim de melhorar a solução, identificar situações não atendidas e verificar o desempenho diário no monitoramento e interação do sistema com as pessoas e o ambiente;
2. para avaliar a aderência do sistema AAL NOCS, será importante uma pesquisa de várias soluções AAL existentes, seguida de uma migração destas soluções para a estrutura do NOCS. Validando sua capacidade de ser extensível, e tornando-o um sistema que ofereça cada vez mais assistência domiciliar aos idosos;
3. para facilitar a parametrização de um ambiente, pretende-se criar de um conceito de *template* de regras, onde o usuário poderá selecionar um modelo de regra para auxiliar na criação de novos ambientes ou novos indivíduos inseridos nos ambientes existentes. O modelo da regra será composto pelos tipos de sensores e atuadores que poderão ser utilizados, além da estrutura inicial da *Rule*, *Conditions*, *Premises*, *Instigations* e *Methods*, tornando ainda mais fácil a tarefa de gerenciamento por profissionais sem conhecimento técnico como médicos, enfermeiros, assistentes sociais, acompanhantes de idosos, monitores ou familiares;
4. atualmente, as leituras dos valores dos sensores são enviadas diretamente para a *FBE,* que por sua vez notifica as *Premises* relacionadas. Com a finalidade de adicionar inteligência artificial ao NOCS, seria importante o desenvolvimento de uma camada que recebe os valores dos sensores e aplica inteligência antes de enviá-los às *FBEs*. Pois, existem regras que devem ser aplicadas onde as condições a serem analisadas são muito mais complexas, por exemplo, *Rules* composta por *Premises* que avaliam desvios no padrão de uso do banheiro, neste caso, a leitura do tempo de cada uso e a

frequência das idas ao banheiro, passariam por uma camada de identificação de padrões, e o resultado desta analise poderia ser utilizada na expressão de uma *Premise* que avalia o percentual de chance de uma incontinência urinária;

5. para um melhor monitoramento e interação da pessoa com o ambiente, pretende-se a implementação total do aplicativo NOCS Mobile para ser executado nos relógios inteligentes (*smartwatches*), inclusive com o monitoramento de passos e batimentos cardíacos no próprio dispositivo, tendo em vista que a utilização da plataforma Xamarin permite um reaproveitamento do código fonte para gerar aplicativos para outros dispositivos, sendo necessário em alguns casos apenas adequações visuais. A Figura 28 apresenta uma ideia de como ficaria o aplicativo NOCS Mobile sendo executado em um relógio inteligente. Dessa forma, é possível deixar a solução ainda mais adaptada às necessidades dos indivíduos inseridos no ambiente monitorado;

Figura 28 – Aplicativo NOCS Mobile sendo executado em relógios inteligentes – *Smartwatches*

Fonte: Autoria própria

6. para uma maior integração entre os dispositivos de um ambiente inteligente, sugere-se que a aplicação NOCS se conecte e interaja com dispositivos de assistência por voz, como, por exemplo, o Alexa da Amazon e o Google Assistant da própria Google;

7. e, finalmente, deseja-se evoluir a solução NOCS, ao ponto de atender múltiplos ambientes com múltiplos indivíduos, respeitando as regras, condições e individualidade de cada pessoa inserida nesses ambientes; dessa forma, a solução poderá ser aplicada em lares de idosos, hospitais, casas de repousos, asilos, condomínios ou até mesmo bairros e cidades inteiras.

REFERÊNCIAS

ACAMPORA, G.; COOK, D. J.; RASHIDI, P.; VASILAKOS, A. V. **A Survey on Ambient Intelligence in Health Care**. Proceedings of the IEEE. Institute of Electrical and Electronics Engineers, v. 101(12), p. 2470–2494, 2013. DOI 10.1109/JPROC.2013.2262913.

ASHTON, K. **That "Internet of Things" Thing**. RFiD Journal, p. 4986, 2009.

BANASZEWSKI, R. F. **"Notification Oriented Paradigm: Advances and Comparisons"**. Original title in Portuguese: **"Paradigma Orientado a Notificações: Avanços e Comparações"**. Master in Science Thesis, Graduate School in Electrical Engineering and Industrial Computer Science (CPGEI) at the Federal University of Technology - Paraná (UTFPR). Curitiba, Paraná, Brazil, March 27, 2009. http://arquivos.cpgei.ct.utfpr.edu.br/Ano_2009/dissertacoes/Dissertacao _500_2009.pdf

BANASZEWSKI, R. F.; SIMÃO, J. M.; TACLA, P. C.; STADZISZ, P. C. **Notification Oriented Paradigm (NOP) - A Software Development Approach based on Artificial Intelligence Concepts.** VI Congress of Logic Applied to Technology – LAPTEC 2007. Santos, 2007.

BARRETO, R. M. W. **Notification Oriented Paradigm in the Context of Distributed Systems**. Relatório da disciplina de Tópicos Especiais Em Ec: Paradigma Orientado A Notificações. CPGEI-PPGCA/UTFPR, Curitiba -PR, Brasil, 2016.

BELMONTE, D.; LINHARES, R. R.; STADZISZ, P. C.; SIMÃO, J. M. **A new Method for Dynamic Balancing of Workload and Scalability in Multicore Systems**. IEEE Latin America Transactions, ISSN: 1548-0992, 2016.

BORKAR, S.; CHIEN, A. A. **The Future of Microprocessors**. Communications of the ACM, v. 54, n. 5, p. 0–5, 2011.

BROOKSHEAR, G**. Computer Science: An Overview**. Addison Wesley, 2012.

CENTER FOR DISEASE CONTROL AND PREVENTION. **The State of Aging and Health in America 2013.** N. A. on an Aging Society, 2013.

CENTER FOR DISEASE CONTROL AND PREVENTION. **National Center for Chronic Disease Prevention and Health Promotion, Division of Population Health. Healthy Aging Data.** Acessado em outubro 2019. URL: https://www.cdc.gov/aging/agingdata/index.html.

COOK, D. J.; DAS, S. K. **How Smart are our Environments? An Updated Look at the State of the Art**. Journal of Pervasive and Mobile Computing, v. 3, p. 53-73, 2007.

EDWING, A.; FERRUZCA, M.; IVAN G. **Detection of Episodes of Major Depression in Older Adults Through Physiological Markers and Movement Patterns Case Study**. 2015 DOI 10.1109/ICHI.2015.48.

ESNAOLA, U.; SMITHERS, T. **"Whistling to Machines," in Ambient Intelligence in Everyday Life**. Lecture Notes in Computer Science Series. v. 3864, New York: Springer, p. 198–226, 2006.

FERREIRA, C. A. **Linguagem e Compilador para o Paradigma Orientado a Notificações (PON): Avanços e Comparações**. Dissertação de Mestrado, PPGCA UTFPR, 2015.

FORKAN, A.; KHALIL, I.; TARI, Z. **CoCaMAAL: A Cloud-Oriented Context-Aware Middleware in Ambient Assisted Living**. Future Generation Computer Systems, v. 35, p. 114–127, 2014. DOI 10.1016/j.future.2013.07.009.

GABBRIELLI, M.; MARTINI, S. **Programming Languages: Principles and Paradigms**. 1st ed. Springer Publishing Company, Incorporated, 2010.

HAIGH, K.; KIFF, L.; MYERS, J.; GURALNIK, V.; GEIB, C.; PHELPS, J.; WAGNER, T. **The Independent Lifestyle Assistant™ (I.L.S.A.): AI Lessons Learned.** P. 852-857, 2004.

HANSEN, S.; FOSSUM, T. V. **Event Based Programming**. Kenosha WI, May 23, 2010. Disponível em: http://www.cs.uwp.edu/staff/hansen/EventsWWW/ e http://www.lulu.com/shop/stuart-hansen/event-driven-programming/ebook/product-17346555.html, 2010.

HENZEN, A. F. **Portabilidade do Framework PON de C++ Standard para C# e Java**. Relatório da disciplina de Tópicos Especiais Em Ec: Paradigma Orientado A Notificações. CPGEI-PPGCA/UTFPR, Curitiba -PR, Brasil, 2015.

ISTAG. **Scenarios for Ambient Intelligence in 2010**. European Commission Report, 2001.

ISTAG. **Strategic Orientations & Priorities for IST in FP6**. European Commission Report, 2002.

KAISLER, S. H. **Software Paradigms.** John Wiley & Sons, 2005.

KATSIRI, E.; BACON, J.; MYCROFT, A. **An Extended Publish/Subscribe Protocol for Transparent Subscriptions to Distributed Abstract State in Sensor-Driven Systems using Abstract Events.** Proc. Int. Workshop on Distributed Event-Based Syst. UK, 2004.

KATSIRI, E.; BACON, J.; MYCROFT, A. **Linking Temporal First-Order Logic and Hidden Markov Models with Abstract Events.** International Journal on Artificial Intelligence Tools, v. 19, p. 857-893, 2010.

KATSIRI, E.; BACON, J.; MYCROFT, A. **Linking Sensor Data to Context-Aware Applications using Abstract Events.** Journal of Pervasive Comp. and Sys., special issue on Managing Context Information in Mobile and Pervasive Environments, p. 3-4, 2007.

KATSIRI, E. **"Middleware Support for Context-Awareness in Distributed Sensor-Driven Systems."** Ph.D. dissertation, University of Cambridge, 2005.

KATSIRI, E.; MYCROFT, A.; BACON, J. **Language and Model-based Abstractions for Estimating Energy Expenditure in Pervasive Healthcare.** Book Chapter - Reasoning in Event-based distributed systems, Springer series on Studies in Comp. Int., 2010.

KATSIRI, E.; MYCROFT, A. **Model Checking for Sentient Computing: An Axiomatic Approach**. Proc. 1st Workshop on Semantics for Mobile Environments, UK, 2005.

KRUGER, C. P.; HANCKE, G. P. **Benchmarking Internet of Things Devices**. 12th IEEE International Conference on Industrial Informatics, INDIN, p. 611-616, 2014.

KWANG-HYUN, P.; ZEUNGNAM, B.; JU-JANG, L.; BYUNG, K.; JONG-TAE, L.; JIN-OH, K.; HEYOUNG, L.; DIMITAR, S.; DAEJIN, K.; JIN-WOO, J.; JUN-HYEONG, D.; KAP-HO, S.; CHONG, K.; WON-GYU, S.; WOO-JUN, L. **Robotic Smart House to Assist People with Movement Disabilities.** Auton. Robots. v. 22. p. 183-198, 2007. DOI 10.1007/s10514-006-9012-9.

LINHARES, R. R.; SIMÃO, J. M.; STADZISZ, P. C. **NOCA – A Notification Oriented Computer Architecture**. IEEE Latin America Transactions, v. 13, Issue 5, May 2015.

LOKE, S. **Context-Aware Pervasive Systems: Architectures for a New Breed of Applications**. CRC Press, 2006.

MOORE, G. E. **Cramming More Components Onto Integrated Circuits**. Electronics Magazine, 1965.

OLIVEIRA, R. N. **Uso do Paradigma Orientado a Notificações em Sistemas Sencientes.** Relatório da disciplina de Tópicos Especiais Em Ec: Paradigma Orientado A Notificações. CPGEI-PPGCA/UTFPR, Curitiba -PR, Brasil, 2016.

OLIVEIRA, R. N.; ROTH, V.; HENZEN, A. F.; SIMÃO, J. M.; WILLE, E. C. G.; NOHAMA, P. **Notification Oriented Paradigm Applied to Ambient Assisted Living Tool**. IEEE Latin America Transactions, 2018.

PORDEUS, L. F. **Simulação de uma Arquitetura de Computação Própria ao Paradigma Orientado a Notificação.** 2017. 364 f. Dissertação de Mestrado. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial (CPGEI). Universidade Tecnológica Federal do Paraná (UTFPR). Curitiba, 2017.

POTDAR, V.; SHARIF, A.; CHANG, E. **"Wireless Sensor Networks: A Survey."** International Conference on Advanced Information Networking and Applications Workshops, v. 38, p. 636–641, 2009.

RAMOS, C. **Ambient Intelligence – A State of the Art from Artificial Intelligence Perspective**. Progress in Artificial Intelligence, v. 18, p. 285-295, 2007.

RASHIDI, P.; MIHAILIDIS, A. **A Survey on Ambient-Assisted Living Tools for Older Adults**. IEEE Journal of Biomedical and Health Informatics, v. 17, n. 3, p. 579-590, 2013.

RONSZCKA, A. F. **Contribuição para a Concepção de Aplicações no Paradigma Orientado a Notificações (PON) sob o viés de Padrões**. Dissertação de mestrado –Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI), UTFPR, 2012.

RONSZCKA, A. F. **Método para a Criação de Linguagens de Programação e Compiladores para o Paradigma Orientado a Notificações em Plataformas Distintas**. 2019. 375f. Tese de Doutorado – Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI). Universidade Tecnológica Federal do Paraná (UTFPR). Curitiba, 2019.

ROY, P. VAN; HARIDI, S. **Concepts, Techniques, and Models of Computer Programming**. Cambridge, MA, USA: MIT Press, 2004.

ROY, P. VAN. **Programming Paradigms for Dummies: What Every Programmer Should Know**. New Computational Paradigms for Computer Music, p. 9–47, 2009.

SANTOS, L. A. **Linguagem e Compilador para o Paradigma Orientado a Notificações: Avanços para Facilitar a Codificação e sua Validação em uma Aplicação de Controle de Futebol de Robôs.** 293 f. Dissertação de Mestrado. Programa de Pós-graduação em Engenharia Elétrica e Informática Empresarial (CPGEI). Universidade Tecnológica Federal do Paraná (UTFPR). Curitiba, 2017.

SCHÜTZ, F.; FABRO, J. A.; RONSZCKA, A. F.; STADZISZ, P. C.; SIMÃO, J. M. **Proposal of a Declarative and Parallelizable Artificial Neural Network Using the Notification-Oriented Paradigm**. Neural Computing and Applications, p. 1-12, 2018.

SCHÜTZ, F. **Neuro-PON: Uma abordagem para o Desenvolvimento de Redes Neurais Artificiais utilizando o Paradigma Orientado a Notificações**. Tese de doutorado–Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI), UTFPR, 2019.

SCOTT, M. L**. Programming Language Pragmatics**. Ed. 4. Elsevier Science & Technology., 2016.

SEBESTYEN, G.; STOICA, I.; HANGAN, A. **Human Activity Recognition and Monitoring for Elderly People**. p. 341-347, 2016. DOI 10.1109/ICCP.2016.7737171.

SILVA, M.; GONCALVES, A.; DANTAS, M.; VANELLI, B.; MANERICHI, G.; SANTOS, S.; FERRANDIN, M.; PINTO, A. **Implementation of IoT for Monitoring Ambient Air in Ubiquitous AAL Environments.** p. 158-161, 2015. DOI 10.1109/SBESC.2015.37.

SIMÃO, J. M. **A Contribution to the Development of a HMS Simulation Tool and Proposition of a Meta-Model for Holonic Control**. Ph. D. Thesis. Graduate School in Electrical Engineering and Industrial Computer Science (CPGEI) at Federal University of Technology - Paraná (UTFPR, Brazil) and Research Center For Automatic Control of Nancy (CRAN) - Henry Poincaré University (UHP, France), 2005. Ph. D Thesis available on: http://arquivos.cpgei.ct.utfpr.edu.br/Ano_2005/teses/Tese_012_2005.pdf.

SIMÃO, J. M.; BANASZEWSKI, R. F.; TACLA, C. A.; STADZISZ, P. C. **Notification Oriented Paradigm (NOP) and Imperative Paradigm: A Comparative Study**. Journal of Software Engineering and Applications (JSEA), p. 402-416, v.5, n.6, 2012. ISSN: 1945-3116. DOI 10.4236/jsea.2012.59083.

SIMÃO, J. M.; RENAUX, D. P. B.; LINHARES, R. R.; STADZISZ, P. C. **Evaluation of the Notification Oriented Paradigm Applied to Sentient Computing**. In: 10th Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2014) in 2014 IEEE 17th International Symposium on Object/Component-Oriented Real-Time Distributed Computing, 2014, Reno - Nevada – USA, 2014. v. 1555-0. p. 253-260. DOI: 10.1109/ISORC.2014.54.

SIMÃO, J. M.; STADZISZ, P. C. **Inference Based on Notifications: A Holonic Meta-Model Applied to Control Issues**. IEEE Transactions on Systems, Man and Cybernetics, Part A. v. 39, Issue 1, Jan. 2009 p. 238-250. DOI 10.1109/TSMCA.2008.2006371.

SIMÃO, J. M.; STADZISZ, P. C. **Notification Oriented Paradigm (NOP) — A Notification Oriented Technique to Software Composition and Execution**. Original title: **Paradigma Orientado a Notificações (PON) Uma Técnica de Composição e Execução de Software Orientada a Notificações**. 2008, Brasil. PEDIDO DE PATENTE: Privilégio de Inovação. Número do registro: PI08055181, data de depósito: 26/11/2008, INPI - Instituto Nacional da Propriedade Industrial. Universidade Tecnológica Federal do Paraná - UTFPR (Demanda Agência de Inovação, 2007). http://www.patentesonline.com.br/paradigma-orientado-a-notificacoespon-uma-tecnica-de-composicao-e-execucao-de-software-234943.html.

SURYADEVARA, N.; MUKHOPADHYAY, S.C. **Determining Wellness through an Ambient Assisted Living Environment.** Intelligent Systems, IEEE, v. 29, p. 30-37, 2014. DOI 10.1109/MIS.2014.16.

TALAU, M. **PONIP: Uso do Paradigma Orientado a Notificações em Redes IP**. Relatório da disciplina de Tópicos Especiais Em Ec: Paradigma Orientado A Notificações. CPGEI-PPGCA/UTFPR, Curitiba -PR, Brasil, 2016.

UNITED NATIONS. **World Population Ageing 2017.** Department of Economic and Social Affairs, Population Division, 2017, ST/ESA/SER.A/408.

VALENÇA, G. Z. **Contribuição para Materialização do Paradigma Orientado a Notificações (PON) Via Framework e Wizard**. Dissertação de Mestrado, Programa de Pós-Graduação em Computação Aplicada (PPGCA), UTFPR. Curitiba, Brasil, 2012.

XAVIER, R. D. **Paradigmas de Desenvolvimento de Software: Comparação entre Abordagens Orientada a Eventos e Orientada a Notificações**, Dissertação de Mestrado, Programa de Pós-Graduação em Computação Aplicada (PPGCA), UTFPR. Curitiba, Brasil, 2014.

WEISER, M. **The Computer for the Twenty-First Century**. Scientific American, pp. 94-104, 1991.