Universidade Federal do Espírito Santo
Centro Tecnológico
Programa de Pós-Graduação em Informática

HealthSimulator: uma plataforma de simulação para desenvolvimento de aplicações de saúde

Renato Nolasco da Rocha

Vitória
2019

Renato Nolasco da Rocha

HealthSimulator: uma plataforma de simulação para desenvolvimento de aplicações de saúde

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Orientador: Prof. Dr. José Gonçalves Pereira Filho

Coorientador: Prof. Dr. Celso Alberto Saibel Santos

Vitória

2019

Ficha catalográfica disponibilizada pelo Sistema Integrado de Bibliotecas - SIBI/UFES e elaborada pelo autor

---

Rocha, Renato Nolasco da, 1992-
R672h HealthSimulator: uma plataforma de simulação para desenvolvimento de aplicações de saúde / Renato Nolasco da Rocha. - 2019.
89 f. : il.

Orientador: José Gonçalves Pereira Filho.
Coorientador: Celso Alberto Saibel Santos.
Dissertação (Mestrado em Informática) - Universidade Federal do Espírito Santo, Centro Tecnológico.

1. IoT. 2. Health. 3. Simulator. I. Pereira Filho, José Gonçalves. II. Santos, Celso Alberto Saibel. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 004

HEALTHSIMULATOR: UMA PLATAFORMA DE SIMULAÇÃO PARA O DESENVOLVIMENTO DE APLICAÇÕES DE SAÚDE

Renato Nolasco da Rocha

Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.

Aprovada em 17 de dezembro de 2019:

Prof. Dr. José Gonçalves Pereira Filho
Orientador(a)

Prof. Dr. Celso Alberto Saibel Santos
Coorientador(a)

Prof. Dr. Rodolfo da Silva Villaca
Membro Interno

Prof. Dr. Clever Ricardo Guareis de Farias
Membro Externo, participação remota

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
Vitória-ES, 17 de dezembro de 2019.

Este trabalho é dedicado à minha família

Agradecimentos

A Deus, o meu guia e protetor. À minha companheira Roselice, sempre disposta a me ajudar nesta jornada, pelo amor, carinho, companheirismo e compreensão. Aos meus pais, Angelo e Creuza, pelo esforço em me educar e incentivo nos momentos cruciais da minha vida. Ao meu irmão Victor por todo o incentivo e conselhos. Ao meu orientador e coorientador, José e Celso, por toda sua colaboração com este trabalho, pela dedicação, confiança e motivação em discutir ideias e materializá-las. Aos colegas do Laboratório de Pesquisas em Redes e Multimídia Jordano, Estevão, Marcello, Sérgio, Saymon, Alessandro, Bruno, dentre outros que não citei, pelas contribuições, pelo convívio e aprendizado. Aos professores do Mestrado em Informática do PPGI/UFES, por compartilhar suas experiências e contribuírem imensamente com meu aprendizado. Ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, por toda infraestrutura disponibilizada durante o curso para o desenvolvimento deste trabalho.

Agradeço também à CAPES e à FAPES pelo apoio financeiro durante a realização deste trabalho.

"Sua maior riqueza é o que você sabe. Seu maior risco é o que você não conhece."
(Robert T. Kiyosaki - Pai Rico, pai pobre)

Resumo

O monitoramento remoto de pacientes (do inglês, *Remote Patient Monitoring* - RPM) em ambiente domiciliar tem sido uma prática comumente empregada no intuito de monitorar a condição de saúde do indivíduo e a conformidade com tratamento proposto pela equipe médica. Além de melhorar a qualidade do atendimento oferecido, esta abordagem tem se provado eficaz na redução das despesas com saúde, uma vez que permite automatizar tarefas outrora realizadas por seres humanos, além de diminuir os episódios de internação e visitas desnecessárias a hospitais e clínicas médicas. Reproduzir cenários reais para testar aplicações de saúde é uma tarefa complexa e desafiadora, particularmente no cenário de RPM, haja vista o número reduzido de bases de dados disponíveis e padronizadas para uso. Uma técnica empregada para realizar testes neste cenário é a simulação dos dados biológicos dos pacientes. Diversas ferramentas têm sido propostas para oferecer suporte esta atividade, porém elas apresentam uma série de lacunas, principalmente relacionadas à escalabilidade e às fontes de dados utilizadas para simulação. Este trabalho apresenta o HealthSimulator, um simulador para avaliação de aplicações de Saúde voltadas a aplicações RPM. O HealthSimulator busca simplificar a simulação de dados biomédicos de pacientes com o intuito fornecer suporte para avaliação de infraestruturas e serviços para aplicações de Saúde. Testes de validação do sistema focados em escalabilidade e emprego de bases de dados reais comprovaram o potencial de aplicação do HealthSimulator para validar aplicações RPM.

Palavras-chaves: Internet das Coisas. Saúde. Simulador.

Abstract

Remote patient monitoring (RPM) in the home environment has been a commonly used practice to monitor an individual's health condition and treatment compliance proposed by the medical team. In addition to improving the quality of care offered, this approach has proven to be effective in reducing health care expenses, as it allows to automate tasks once performed by humans, as well as reducing episodes of hospitalization and unnecessary visits to hospitals and medical clinics. Reproducing real-world scenarios for testing health applications is a complex and challenging task, particularly in the RPM scenario, given the small number of open and standardized databases available for use. One technique employed to perform tests in this scenario is the simulation of the biological patient's data. Several tools have been proposed to support this, but they have several gaps, mainly related to scalability and data sources used for simulation. This paper presents the HealthSimulator, a simulator for evaluation of Health applications focused on RPM applications. HealthSimulator seeks to simplify the simulation of biomedical patient data to provide support for evaluating infrastructure and services for Health applications. System validation tests focused on scalability, and real-world database use have proven the potential for HealthSimulator application to validate RPM applications.

Keywords: Internet of Things. Health. Simulator.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AAL	Assistência à Autonomia no Domicílio
$CoAP$	Constrained Application Protocol
CSV	Comma Separated Values
IoT	Internet of Things
$JSON$	JavaScript Object Notation
$HTTP$	Hypertext Transfer Protocol
$HL7$	Health Level Seven
$MQTT$	Message Queue Telemetry Transport
$REST$	Representational Ttate Transfer
TCP	Transmission Control Protocol
UDP	User Data Gramprotocol
XML	Extensible Markup Language
$YAML$	YAML Ain't Markup Language

Sumário

1 Introdução

Recentes avanços nas tecnologias de integração de componentes eletrônicos permitiram o surgimento de sistemas computacionais miniaturizados de precisão, os chamados *Microelectromechanical Systems* (MEMS). Estes sistemas impulsionaram o desenvolvimento de novos dispositivos e plataformas de *hardware* de sensoriamento, processamento e comunicação sem fio, que formaram a base para criação de um novo conceito de *internet*, a "Internet das Coisas" (do inglês, *Internet of Things* – IoT) (RAJA; RAJKUMAR; RAJ, 2018; SANTOS et al., 2016; Ud Din et al., 2019).

IoT é um cenário de *Internet* do Futuro que se caracteriza pela visão de uma rede formada por objetos inteligentes (*smart objects*) interconectados e endereçáveis, a partir do uso e adaptação de protocolos e padrões de comunicação bem definidos e conhecidos da *Internet* atual. Neste novo cenário, objetos do mundo real, isto é, coisas (*things*) do cotidiano, tais como ferramentas, eletrodomésticos, itens pessoais e de vestuário etc, são embarcados com capacidades de processamento, armazenamento, sensoriamento e comunicação sem fio, e integrados à *Internet*. Estes objetos ou coisas são vistos pelas aplicações como entidades autônomas, com comportamento pró-ativo e conhecimento sobre o contexto circundante, além de capacidade de colaboração para alcançar uma meta comum.

A *Internet* das Coisas vem entrando fortemente no dia-a-dia das pessoas e em diferentes setores da economia. É possível perceber que praticamente todos os setores da sociedade estão sendo ou podem ser remodelados com a ajuda da conectividade e do acesso às informações em tempo real que a IoT proporciona. Isto é notável, por exemplo, quando observamos os avanços e as tendências tecnológicas na área da Saúde (RGHIOUI et al., 2017).

A área da Saúde, que constitui o domínio geral de aplicação deste trabalho, é uma questão de interesse global. Prevê-se que os gastos mundiais em Saúde excedam US$ 18 trilhões até 2040 (Institute for Health Metrics and Evaluation, 2016), o que mostra a relevância da área e a importância que as soluções de IoT terão num futuro próximo. O setor de Saúde permite inúmeras possibilidades de aplicação das tecnologias de IoT, por exemplo, na prevenção, acompanhamento de doenças crônicas e na redução de infecções hospitalares, fatores que representam os principais agravantes em relação à piora da qualidade de vida dos pacientes e dos altos custos das instituições de Saúde (ISLAM et al., 2015).

A literatura recente de *Internet das Coisas* e Informática aplicada à Saúde conduz relatos de aplicações IoT que trouxeram importantes benefícios para pacientes, pesquisa-

dores, unidades e profissionais de Saúde, por exemplo, (BLEDA et al., 2013; MEMON et al., 2014; SU et al., 2019; CELESTRINI et al., 2019; SURAKI; JAHANSHAHI, 2013; PALATTELLA et al., 2016; DEWEY; DEVRIES, 2018; Tsiachri Renta; SOTIRIADIS; PETRAKIS, 2017). De fato, aplicações IoT podem proporcionar melhorias significativas nos serviços médicos e de assistência à Saúde, incluindo a redução do tempo de espera nas salas de emergência, a garantia da disponibilidade e acessibilidade de *hardware* crítico, o auxílio ao tratamento de doenças crônicas, o rastreamento de equipes, pacientes e inventários e, ainda, aprimoramento do gerenciamento de medicamentos, entre outros benefícios (KAYLA, 2018). Tecnologias de IoT também podem contribuir para a implementação de soluções inovadoras de telemonitoramento, como os sistemas *Ambient Assisted Living* (AAL), os quais permitem o monitoramento e acompanhamento remoto de pacientes em conformidade com o tratamento daqueles que preferem ou que precisam ficar em casa por alguma limitação. Essas aplicações implementam funcionalidades para o monitoramento contínuo dos pacientes, com facilidades de acesso às informações sobre estado de saúde, histórico médico completo para apoio a diagnósticos assertivos, facilidades para o compartilhamento de informações, além de auxiliar nas ações preventivas e de auto-cuidado.

Vislumbra-se, num futuro próximo, um cenário onde seja possível proporcionar serviços de monitoramento e acompanhamento em ambientes inteligentes em escala global. Entretanto, um serviço de Saúde sensível como este acaba se tornando um grande desafio para as aplicações de eHealth devido ao número enorme de pacientes e à quantidade gigantesca de dados gerados pelas suas fontes, i.e., pelos dispositivos IoT, tornando desafiador o processo de coleta e manipulação deste volume massivo de dados biomédicos.

Assim, um obstáculo particularmente difícil é reproduzir estes cenários reais de grande escala, com vistas aos testes das aplicações. Existem inúmeras dificuldades para se testar aplicações de monitoramento em Saúde, mas certamente uma das principais é esta complexidade de se copiar o ambiente real em que elas operarão, não apenas pela dificuldade de escala mas, também, pela inviabilidade de se utilizar dados reais dos pacientes devido a questões de privacidade e de legislação. Outras questões menos discutidas, porém importantes, estão ligadas às interfaces inadequadas, à diversidade de formatos de dados, aos protocolos de comunicação não padronizados etc., que complementam os principais desafios dos sistemas atuais de monitoramento em Saúde. A falta de tratamento destes problemas pode resultar em soluções de baixa precisão ou mesmo inviabilizar algumas delas.

Assim, em face das dificuldades apontadas, os desenvolvedores de sistemas eHealth se utilizam de ferramentas de software que tentam reproduzir - e testar - o cenário de interesse, fazendo uso de *Simulação*. De fato, a reprodução dos processos que ocorrem em um ambiente real de cuidados à Saúde, realizada de forma não exploratória, é inadequada

para a percepção ou busca de falhas no progresso do desenvolvimento das soluções computacionais de Saúde. Nesse sentido, ambientes de testes que reproduzem o cenário de interesse possibilitam que os processos sejam desenvolvidos dentro de um ambiente controlado, fazendo com que se investigue uma grande gama de situações possíveis de condições de falha da aplicação. A exploração proporcionada por meio das ferramentas de simulação permite a identificação de possíveis problemas que ocorrem ao longo do tempo no ambiente real, possibilitando a correção de eventuais falhas de maneira antecipada.

Uma revisão da literatura verificou a existência de ferramentas de simulação, algumas dos quais voltadas para o domínio de IoT em Saúde (BRAY et al., 2019; EDELMANN et al., 2018; CIUCU; METHODS, 2015; LO, 2015; BACHLER et al., 2015; BERHANU; ABIE; HAMDI, 2013; LO et al., 2012; AGAR; EREN; CINAR, 2006). Tais simuladores, entretanto, apresentam importantes lacunas funcionais na sua arquitetura no que se refere às seguintes questões:

(I) falta de utilização de dados biomédicos reais coletados de diferentes repositórios de dados de pesquisa médicas disponíveis, formados por registros de indivíduos saudáveis e pacientes com uma variedade de condições com implicações importantes na Saúde Pública;

(II) pouca preocupação com a questão da interoperabilidade entre os simuladores e as aplicações de Saúde, o que significa que os simuladores não apresentam um padrão de comunicação comum utilizado pela maioria das aplicações de Saúde, tais como o HL7 (*Health Level* 7) ou FHIR (*Fast Healthcare Interoperability Resources*), que são amplamente implementados pela indústria da Saúde (BENDER; SARTIPI, 2013);

(III) pouco foco no tratamento de cenários de alta escalabilidade das aplicações, não garantindo o envio de grandes volumes de registros (reais) de pacientes para as aplicações sob teste;

(IV) baixa preocupação com a interface humano-computador, sem garantias de que esta promova a visualização da simulação de forma simples e gráfica, de modo a facilitar a análise dos dados e a implantação de medidas corretivas.

1.1 Objetivos

Considerando o cenário acima descrito, este trabalho tem por objetivo desenvolver um simulador IoT cuja arquitetura conceitual incorpore facilidades de acesso e coleta de registros biomédicos localizados em repositórios de dados abertos reais, e a formatação e disponibilização destes dados para as aplicações eHealth através do uso de padrões de dados e de comunicação, com vistas a auxiliar os testes e a análise de funcionalidades e escalabilidade de aplicações e serviços na área de Saúde.

1.2 Metodologia

A metodologia empregada no desenvolvimento do trabalho incluiu, inicialmente, um estudo do domínio de IoT e de suas aplicações da área de Saúde, o que compõe as linhas de pesquisa principais desta dissertação. Numa etapa seguinte, diversas ferramentas de simulação foram pesquisadas e comparadas em relação aos seguintes critérios: camada de atuação, escopo (acadêmico ou corporativo), ambiente de testes e de execução, tipo de linguagem de desenvolvimento, tipo de licença e escalabilidade. Esta pesquisa serviu de base para a escolha de trabalhos de referência para o desenvolvimento da proposta. O conjunto de funcionalidades apresentadas, a opção por código aberto, o uso de linguagem de programação adequada, a facilidade de uso e a boa aceitação por parte da comunidade acadêmica, foram usados como critérios de seleção. Ao mesmo tempo, foi possível identificar nesta análise os principais desafios e problemas em aberto. Também foram identificadas quais estratégias poderiam ser generalizadas para uso em outras aplicações similares. Por fim, foram determinados quais componentes arquiteturais são essenciais, quais são desejáveis e quais são opcionais numa plataforma que visa a rápida prototipagem das aplicações. Esta análise detalhada permitiu definir a proposta final de arquitetura da solução de simulação e a implementação do simulador de dados de saúde apresentada no Capítulo 4, que constitui a principal contribuição deste trabalho.

Para atingir os objetivos o trabalho foi dividido nas seguintes etapas:

- estudo da literatura com ênfase nos simuladores IoT para ambientes de Saúde como forma de conhecer as soluções mais recentes e levantar os principais requisitos que regem a construção desta classe de sistemas;
- concepção de uma arquitetura conceitual que incorpore componentes e funcionalidades específicas para acesso e coleta de registros biomédicos localizados em fontes externas e disponibilização destes dados para uso por aplicações de Saúde;
- implementação desta arquitetura com base em uma infraestrutura tecnológica que promova a escalabilidade, adote padrões de comunicação e facilite o acesso a registros biomédicos reais de repositórios online, já validados pela comunidade de Saúde;
- desenvolver uma aplicação eHealth como forma de demonstrar as funcionalidades da infraestrutura de simulação desenvolvida.

1.3 Organização da Dissertação

O restante desta dissertação está estruturada da seguinte forma:

- O Capítulo 2 apresenta a fundamentação teórica do trabalho.

- O Capítulo 3 apresenta uma pesquisa exploratória de ferramentas de simulação IoT. É feito um levantamento das ferramentas mais utilizadas, as quais são comparadas com vistas ao levantamento de requisitos e a definição das características desejadas na ferramenta de simulação proposta.

- O Capítulo 4 introduz a arquitetura da plataforma de simulação proposta no trabalho, tendo por base os requisitos observados.

- O Capítulo 5 apresenta a implementação da plataforma bem como os testes de validação da infraestrutura desenvolvida. Dois testes são mostrados, um deles simulando dados gerados em um processo de acompanhamento de paciente em domicílio, e o outro demonstrando a escalabilidade do simulador.

- O Capítulo 6 conclui a dissertação, apresentando as considerações finais e as perspectivas de trabalhos futuros.

2 Cenário de Aplicação e Tecnologias Aplicadas à Saúde

Este capítulo apresenta uma visão geral do domínio de aplicação deste trabalho, isto é, o ambiente de cuidados com a Saúde em domicílio. O objetivo é identificar características particulares deste ambiente que possam guiar a construção de sistemas de monitoramento remoto de pacientes, bem como vislumbrar funcionalidades necessárias na infraestrutura de simulação proposta. O capítulo trata também de tecnologias aplicadas à Saúde que são usadas no desenvolvimento de aplicações e plataformas como a proposta nesta dissertação. São introduzidos conceitos de repositórios de dados biomédicos, padrões de informações em Saúde, formatos de dados para troca de informações e interoperabilidade em Saúde. Técnicas de virtualização, importantes para tratar problemas que envolvem a escalabilidade de sistemas - um desafio do cenário escolhido - bem como protocolos de comunicação com dispositivos IoT e o paradigma de programação orientada a fluxo de dados, empregado para desenvolver a prova de conceito, são também abordados no capítulo.

2.1 O Cenário de aplicação

A disponibilidade de novos dispositivos inteligentes de monitoramento de pacientes e o aumento dos custos dos cuidados com a Saúde, em especial os altos custos de internação, podem ser considerados os principais ingredientes da introdução da *Internet* das Coisas no ambiente de prevenção e cuidados médicos. De fato, conforme o tipo de doença e o protocolo de tratamento, é possível que o paciente permaneça por longo tempo internado em um hospital, encarecendo o custo de internação devido à falta de infraestrutura adequada para o acompanhamento da evolução da sua saúde no seu retorno ao domicílio. Além disso, segundo (RGHIOUI et al., 2017), a disponibilidade de novas tecnologias de monitoramento dos sinais fisiológicos dos pacientes fora do ambiente hospitalar, permite aos médicos e profissionais de Saúde responderem muito mais rapidamente às situações de emergência médica (MCCONALOGUE; DAVIS; CONNOLLY, 2018). O monitoramento em domicílio também pode reduzir a necessidade de pacientes crônicos (portadores de doenças que não são resolvidas em um prazo curto) de deslocar-se com frequência até o consultório médico.

O monitoramento em domicílio visa, portanto, à melhoria da qualidade do atendimento e a redução com as despesas de saúde por intermédio da automação de tarefas realizadas outrora por seres humanos, durante 24 horas por dia, 7 dias por semana (SU et al., 2019). O monitoramento domiciliar do paciente, especialmente de pessoas idosas com necessidades especiais ou doenças crônicas, como diabetes e insuficiência cardíaca, é, por

conseguinte, uma aplicação de grande relevância e impacto social. Nos países europeus, por exemplo, onde as melhorias no campo da Medicina estão gerando mudanças no estilo de vida e aumentando a expectativa de vida da população, a Comissão Europeia estima que em 2025 mais de 20% dos europeus terão 65 anos ou mais, com um aumento particularmente rápido no número de pessoas com mais de 80 anos (RGHIOUI et al., 2017). Este cenário de monitoramento e assistência médica em domicílio compõe o contexto de aplicação deste trabalho. Em particular, o ambiente de assistência no domicílio usado como referência para o trabalho - e introduzido nas seções a seguir - foi idealizado com base no estudo de trabalhos correlatos encontrados na literatura e na infraestrutura proposta em uma pesquisa de doutorado em andamento do Laboratório de Pesquisa de Redes e Multimídia (LPRM), na área de Informática em Saúde.

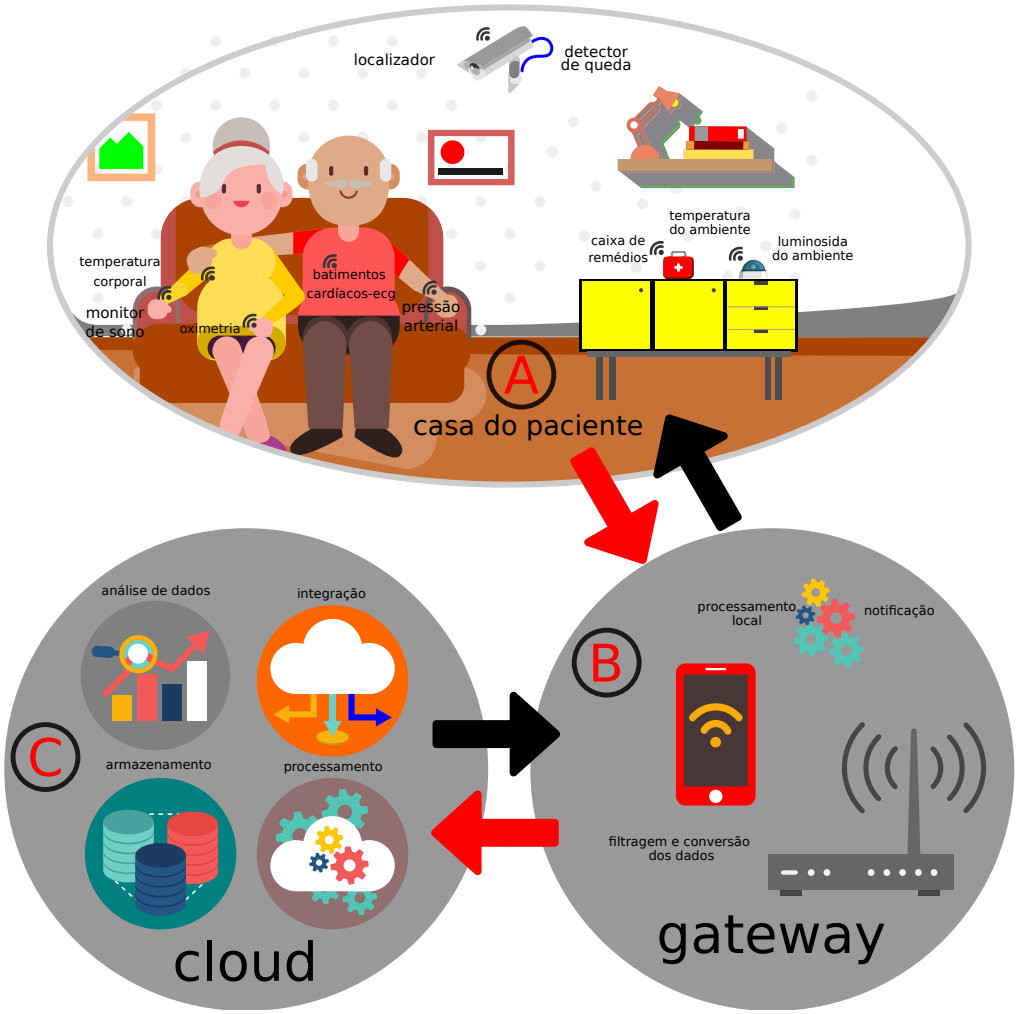
2.2 O Ambiente de assistência no domicílio

A nossa visão de cenário de assistência à autonomia de pacientes no domicílio, incorpora o uso de dispositivos de IoT e apresenta as seguintes etapas: sensoriamento, comunicação e processamento dos dados coletados (MSHALI et al., 2018). Estas três etapas são representadas na Figura 1 pelos itens (A), (B) e (C).

Os sensores no domicílio dos pacientes encontram-se distribuídos espacialmente para monitorar fenômenos físicos ou ambientais. Esses dispositivos podem ser divididos em duas categorias: os incorporados ao ambiente e os dispositivos tecnológicos que podem ser utilizados pelo paciente como peça do vestuário (Figura 1(A)). Os vários sensores utilizados para monitorar os sinais vitais dos pacientes e variáveis do ambiente, de modo contínuo ou periódico, estão embarcados em equipamentos médicos e dispositivos de medição de frequência cardíaca, saturação de oxigênio, temperatura corporal, pressão arterial, glicemia, monitores de sono, câmeras e medidores de luminosidade do ambiente, dentre outros. Sensores adicionais podem ser utilizados neste cenário, tais como sensores de eletrocardiograma (ECG), usados para monitorar a atividade cardíaca, sensores eletroencefalográficos (EEG), para monitorar a atividade cerebral, sensores eletromiográficos (EMG), para monitorar a atividade muscular, e sensores eletrooculográficos (EOG), usados para monitorar movimentos oculares (MSHALI et al., 2018).

Às vezes, simplesmente tomando medidas profiláticas mediante a análise e interpretação dos registros médicos coletados por essa diversidade de sensores e dispositivos de monitoramento, consegue-se uma redução significativa do número de internações recorrentes dos pacientes crônicos. Os sinais fisiológicos e registros médicos coletados também podem ser usados para abastecer prontuários eletrônicos em hospitais ou clínicas médicas, e ainda fornecer suporte à tomada de decisões. Após a coleta, os dados são enviados para um servidor na nuvem utilizando as tecnologias de rede disponíveis ou podem ser

Figura 1 – Visão geral do cenário de aplicação



Fonte: Elaborado pelo autor

processados por meio de um *Gateway* que esteja localizado próximo aos dispositivos. O componente *Gateway* é um elemento regular em ambientes IoT, e usualmente é visto como o coordenador de rede que fornece controle de acesso remoto, ou seja, ele atua como uma ponte entre a rede local e a *Internet*. Existem situações em que há necessidade de utilizar um *Gateway* mais robusto, com capacidade de preprocessar ou processar dados mais próximo à fonte de dados, como é o caso de monitoramento de doenças graves que demandam um menor tempo de resposta e latência da rede. Em certas situações, dispositivos portáteis como *Smartphones*, *Raspberry*, além de outros dispositivos, podem atuar como *Gateways*, e até mesmo como um pequeno servidor, armazenando os dados por um determinado período, como apresentado no item (B) da Figura 1.

No nosso cenário, o componente *Cloud* é responsável por gerenciar os dados da aplicação de monitoramento. Este componente, representado na Figura 1 pelo item (C), é

utilizado para organizar e processar de forma inteligente os dados massivos e por executar a agregação de dados de diferentes fontes. O componente *Cloud* também é encarregado pela interatividade da aplicação e por estabelecer a conexão com o usuário (pacientes, familiares ou profissionais de saúde) do sistema de monitoramento através da exibição das informações por meio de uma interface gráfica. Este componente apresenta quatro módulos principais: `análise de dados`, `armazenamento`, `processamento` e ainda a `integração` de outros serviços. De acordo com as características apresentadas pelo sistema, as interfaces gráficas podem ser utilizadas para monitorar e também por avaliar as condições de saúde mediante a `análise dos dados`. Caso ocorra alguma anormalidade ou situação de emergência com o paciente, esse componente poderá notificar o administrador do sistema de saúde e ainda armazenar as informações relevantes para uma possível apuração do evento.

Ainda no componente *Cloud* o módulo de `processamento` é responsável pela extração, processamento e pelo envio das informações geradas e capturadas a um servidor de banco de dados. Neste módulo, os dados provenientes de sensores heterogêneos podem ser organizados para análise subsequente, em contexto com as variáveis do ambiente (por exemplo, temperatura, umidade relativa do ar e detecção de som do ambiente), rastreamento de movimento, localização e ainda os sinais vitais do paciente (por exemplo, nível de oxigênio no sangue e frequência cardíaca).

Outro módulo presente no componente *Cloud* é o de `análise de dados`, responsável pela atividade de transformar um conjunto de dados brutos do paciente e do ambiente em informação, ou seja, realizar uma análise racional dos dados. Por meio desse módulo é possível analisar dados e identificar alguns sintomas físicos que médicos experientes, possivelmente não conseguiriam encontrar as causas aparentes e correlacioná-los com os problemas identificados no paciente.

O módulo de `armazenamento` gerencia uma coleção de informações, tabelas de dados e outros objetos que são organizados e apresentados para servir um propósito específico, com as facilidades de pesquisa, classificação e combinação de dados. Em aplicações de saúde é essencial que os médicos tenham acesso aos dados do paciente, quando necessário, e que os vários sistemas de saúde trabalhem em conjunto para garantir o perfeito funcionamento.

Diferentemente do módulo de armazenamento, que busca gerenciar o banco de dados local, o módulo de `integração` visa propor uma solução para a gestão da informação de múltiplas fontes de dados. Assim, o seu propósito é apoiar a obtenção e integração de diferentes bases de dados de Saúde, considerando a existência de sistemas de Saúde com propósitos diferentes. Desta forma ele facilita a construção de consultas analíticas e a navegação nas informações resultantes, possibilitando a geração de relatórios integrados, a apresentação de resultados em forma de gráficos e a mineração de dados.

Por fim, vale mencionar que as soluções de Saúde existentes podem ser diferenciadas com base em três aspectos distintos: o alvo, os recursos e os métodos empregados. Em

primeiro lugar, dependendo da população alvo, os projetos da área de Saúde levam em consideração as necessidades e os requisitos do indivíduo, o que pode delimitar o contexto da proposta a um único parâmetro ou objetivo a ser observado. A título de exemplo, existem na literatura projetos que focam exclusivamente no monitoramento para detecção de queda do paciente (MUBASHIR; SHAO; SEED, 2013; KOKALKI et al., 2018). Do mesmo modo, há projetos que avaliam um único tipo de atividade diária como o sedentarismo, uma doença biológica ou um determinado tipo de doença cognitiva (acidente vascular cerebral) em virtude das restrições encontradas no ambiente (ZAMANIFAR; NAZEMI, 2019; HOSSAIN et al., 2017). Em segundo lugar, aspectos referentes à capacidade do sistema está correlacionada ao seu modo de comunicação, processamento, ao seu esquema de armazenamento, à largura de banda disponível e ao gerenciamento de energia dos dispositivos empregados no ambiente. Um aspecto final a ser observado é o método ou a abordagem empregada no desenvolvimento da solução, onde é necessário considerar as limitações existentes devido aos recursos disponíveis nos sistemas inteligentes de Saúde.

2.3 Repositório de dados biomédicos

Um repositório de dados pode ser definido como um ambiente digital desenvolvido para auxiliar pesquisadores no gerenciamento, disponibilização e acesso a dados, contribuindo deste modo com a sua reutilização (CRISTINA et al., 2017). Repositórios viabilizam o armazenamento e facilitam a representação, a disseminação e a preservação dos dados neles depositados. No caso da área da Saúde, o agrupamento de dados biomédicos em repositórios propicia o compartilhamento e promove o reuso entre profissionais da área, pesquisadores e desenvolvedores de aplicações no domínio da Saúde. Um repositório de dados biomédicos muitas vezes fornece conjuntos de dados de diferentes sinais fisiológicos, nos mais variados formatos, que podem ser usados para apoiar a pesquisa e o desenvolvimento de soluções de cuidados com a saúde. Um exemplo importante de repositório e rede de intercâmbio de dados de saúde é o PhysioNet (GOLDBERGER et al., 2000).

O PhysioNet, PhyisioBank e PhysioToolkit são três componentes interdependentes da iniciativa *"Research Resource for Complex Physiologic Signals"*, criada pelo *National Center for Research Resources of the National Institutes of Health*, dos Estados Unidos, que visa disseminar a pesquisa médica por meio do acesso e compartilhamento de dados biomédicos complexos e estimular o desenvolvimento de novos algoritmos e soluções na área de Saúde. A iniciativa tem por base o acesso online e gratuito a repositórios de dados com uma variedade de sinais fisiológicos e outros dados clínicos, por exemplo, sinais de ECG e imagens de tomografia computadorizada intracranial (REYNOLDS; CAHILL; SENART, 2006).

Embora comumente usado para referenciar o conjunto formado pelos três compo-

nentes e mesmo a própria base de dados, o PhysioNet é, na verdade, o componente que implementa um fórum online para a disseminação e troca de sinais biomédicos gravados e *software* de código aberto para analisá-los. Para tal, o Physionet fornece facilidades para a análise cooperativa de dados e a avaliação dos novos algoritmos propostos. Além disso, ele fornece acesso Web gratuito aos dados do PhysioBank, a base de dados propriamente dita, e ao *software* PhysioToolkit (http://www.physionet.org). O PhysioNet também oferece serviços e treinamento através de tutoriais online para ajudar os usuários com diferentes níveis de conhecimento. Devido a sua simplicidade o PhysioNet é muito utilizado para testar e validar diversas soluções de Saúde (CIUCU; METHODS, 2015; ZAMANIFAR; NAZEMI, 2019; EDELMANN et al., 2018).

O banco de dados PhysioBank é um grande e crescente arquivo de gravações digitais de sinais fisiológicos e dados relacionados para uso pela comunidade de pesquisa biomédica, abastecido com registros coletados a partir de uma ampla gama de estudos, desenvolvidos e contribuídos por membros da própria comunidade de pesquisa (MOODY; MARK; GOLDBERGER, 2011). O PhysioBank inclui atualmente mais de 50 coleções de sinais cardiopulmonares, neurais e outros dados biomédicos de indivíduos saudáveis e pacientes com uma variedade de condições com importantes implicações para a saúde pública, incluindo arritmias com risco de vida, insuficiência cardíaca congestiva, epilepsia, apneia do sono e envelhecimento (TANTAWI et al., 2012).

Para facilitar a utilização dos repositórios por parte dos desenvolvedores foi desenvolvido um conjunto de ferramentas, denominado PhysioToolkit, que é composto por uma biblioteca de *software* para processamento e análise de sinais fisiológicos, detecção de eventos empregando técnicas clássicas e novos métodos estatísticos (MOODY; MARK; GOLDBERGER, 2001). Além disso, também permite a exibição, a caracterização interativa de sinais e a criação de novas bases de dados com vistas à simulação de sinais fisiológicos.

Dessa forma, devido à conveniência oferecida pelo PhysioNet e outros repositórios de dados biomédicos, pesquisadores e desenvolvedores podem atualmente fazer *download* de variadas coleções de registros, com recursos personalizados e específicos (tipo de dados), que podem ser adequados a seus interesses particulares para realizar diferentes estudos, validação e testes de aplicações de Saúde por meio de simulações. Infelizmente, a funcionalidade de acesso a repositórios de dados biométricos reais de pacientes, como disponibilizado pelo PhysioNet e outros repositórios da área médica, ainda é limitada nas ferramentas de simulação IoT. Este fato é explorado pelo ambiente proposto nesta dissertação.

2.4 Padrões de informação em Saúde

Há algumas décadas havia pouca necessidade das organizações de Saúde e seus respectivos sistemas de gerenciamento em Saúde trocarem informações administrativas e de registros de pacientes. Porém, o crescimento populacional, a maior expectativa de vida, o surgimento de novas doenças, e a alta incidência e prevalência das doenças crônicas, passaram a exigir modelos organizacionais e sistemas computacionais de atenção à Saúde que sejam dinâmicos e interoperáveis. Esta necessidade de compartilhamento e troca de dados e informações clínicas é justificada por vários fatores, dentre eles a contenção de despesas, a melhoria na atenção e segurança do paciente, a coordenação de cuidados e reconciliações medicamentosas, e a eficiência e eficácia dos resultados das instituições, bem como a melhor formulação de políticas públicas. Neste cenário, a exigência por um registro eletrônico de saúde centralizado é apenas uma dentre as várias demandas por padrões de informação em Saúde que promovam a interoperabilidade.

No Brasil, existe uma regulamentação a respeito da utilização dos padrões de informação em Saúde e interoperabilidade, a Portaria no. 2.073 de agosto de 2011 (Ministério da Saúde, 2011). De acordo com (MIYOSHI, 2018), esta portaria orienta quanto aos padrões de informação a serem utilizados, por exemplo:

- HL7, que foca na interoperabilidade entre sistemas com vistas à integração dos resultados e solicitações de exames;
- OpenEHR, para definição do Registro Eletrônico em Saúde (RES);
- DICOM, para troca de exames de imagem;
- LOINC, para codificação de exames laboratoriais;
- TISS, para interoperabilidade de sistemas de saúde suplementar;
- HL7 CDA, para definição da arquitetura do documento clínico;
- ISBT 128, para identificação das etiquetas de biomateriais;
- IHE-PIX (*Patient Identifier Cross Referencing*) para cruzamento de pacientes de diferentes sistemas de informação;
- SNOMED-CT, para codificação de termos clínicos e mapeamento de terminologias nacionais e internacionais;
- ISO 13606-2 para interoperabilidade de modelos de conhecimento, incluindo arquétipos, templates e metodologia de gestão.

Um relatório realizado em 2016 pela *Global System for Mobile Communications* (GSMA), uma associação de operadoras móveis, avalia os padrões globais existentes e fornece recomendações sobre como a interoperabilidade pode ser realizada ao implementar os serviços de eHealth (GSMA, 2016). Este relatório discute sobre os seguintes padrões: (I) EDIFaCT – *Electronic Data Interchange for Administration*, um padrão de mensagens para o comércio e transporte; (II) Ehr – *Electronic Health Record*, um termo utilizado para descrever um registro de saúde armazenado de forma eletrônica; (III) IEEE 11073, um padrão de codificação para definir leituras de dispositivos médicos; (IV) HL7 FHIR – *Fast Healthcare Interoperability Resources*, um padrão para a troca eletrônica de informações sobre saúde, que define um conjunto de "Recursos" que representam conceitos clínicos granulares. Os recursos podem ser gerenciados isoladamente ou agregados em documentos complexos nos formatos XML, JSON, HTTP e outros.

Como pode ser visto, existem diversos padrões e recomendações de formatos a serem utilizados na comunicação de sistemas de Saúde. De particular interesse para esta dissertação está o padrão HL7, em particular o HL7 FHIR, que é o mais novo padrão desenvolvido pela HL7 *International* para o âmbito da Saúde. O HL7 FHIR, detalhado adiante, se destaca pela sua facilidade de implementação, documentação e por utilizar tecnologia *Web* para compartilhar o próprio padrão. Além disso, o padrão FHIR busca facilitar o seu uso por meio da incorporação do significado dos dados clínicos no padrão (metadados), de modo que possa ser lido por humanos e por máquinas.

O HL7 é atualmente um dos padrões mais utilizados pela indústria e pelos principais atores de TIC em Saúde (fabricantes de equipamentos, hospitais, clínicas, laboratórios, farmácias, prestadores de serviços de Saúde, etc) para a troca de dados entre sistemas, aplicativos e equipamentos médicos. O HL7 fornece uma interface padrão que é utilizada pela indústria da Saúde e por prestadores de serviços e desenvolvedores de soluções de TIC em Saúde, em todo o mundo, como modelo de referência para interoperar dados. Por exemplo, com o uso de um motor de interface HL7, é possível conectar-se a sistemas externos, por exemplo, serviços terceirizados de radiologia, laboratórios, e centros de distribuição de material e medicação hospitalar. O HL7 foi elaborado pela *Health Level Seven International*, uma organização internacional de desenvolvimento de padrões, e é aceito por outras organizações de desenvolvimento de padrões, como o *American National Standards Institute* (ANSI) e a *International Organization for Standardization* (ISO)(RODRIGUES, 2010).

O HL7 fornece a infraestrutura para a integração, o compartilhamento e a recuperação de informações de registros eletrônicos em Saúde, sendo formado por um conjunto de padrões específicos que definem mensagens para praticamente todas as áreas de atenção à Saúde. Por exemplo, ele define quais itens de dados devem ser trocados quando ocorrem determinados eventos clínicos como a admissão, a alta ou a transferência de um paciente (LENZ; REICHERT, 2007). Mensagens HL7 são definidas para (LIU et al., 2019): Admissão

(*Registration*); Controle de documentos (*Document Control*); Ordens (clínica e outras) (Orders); Agendamento e logística (*Scheduling and logistics*); Resultados e Observações (*Results and Observations*); Administração de pessoal (*Personnel administration*); Consultas (*Querys*); Planejamento de cuidados ao paciente (*Patient Care Planing*); Finanças (Finances); Sincronização de rede (*Network Syncronization*); Arquivos mestre e índices (*Master files and indexes*); e Automação de laboratório (*Laboratory automation*).

Os padrões HL7 atuam na camada de aplicação, que é a camada 7 do modelo ISO-OSI. A HL7 considera que dos seus padrões os mais utilizados e implementados são:

- GELLO - uma linguagem de expressão padrão usada para apoio à decisão clínica;
- *Sintaxe Arden* - uma gramática para representar as condições, decisões, recomendações, alertas, lembretes, etc., em sistemas de decisão, denominados de Módulos de Lógica Médica (MLM: *Medical Logical Modules*);
- *Structured Product Labeling* (HL7 SPL: Rotulagem Estruturada de Produtos) - padrão que define as informações publicadas que acompanham um medicamento, com base em HL7 versão 3
- *Clinical Context Object Workgroup* (CCOW) - uma especificação de interoperabilidade para a integração visual dos aplicativos do usuário;
- Anexos de Faturamento (*Claims Annex*) - um anexo padrão de saúde que serve para expandir uma transação financeira na área de saúde;
- Padrão de Mensagens HL7 Versão 2.x - uma especificação de interoperabilidade para transações médicas e de saúde;
- Padrão de Mensagens HL7 Versão 3 - uma especificação de interoperabilidade para as transações médicas e de saúde, com base no RIM;
- *Clinical Document Architecture* (HL CDA) - um modelo de troca de documentos clínicos, com base no HL7 versão 3 e no RIM;
- *Fast Interoperable Health Resources* (FHIR: Recursos Interoperáveis Rápidos em Saúde) - um novo conjunto de padrões apoiados em modelos da *Web*, mais compacto e fácil de usar do que o HL7 V3 e o CDA, e que supostamente irá substituí-los no futuro.

Em suma, o HL7 permite a implantação e a integração de todo ambiente de Saúde, desde pacientes, hospitais, médicos e operadoras de planos de saúde, de forma facilitada (GSMA, 2016). Por intermédio do uso da estrutura proposta pelo HL7, pode ocorrer a redução do agravamento do quadro clínico do paciente, mediante ao acesso às

informações completas para um diagnóstico mais assertivo. Por fim, as interfaces HL7 também promovem um melhor o fluxo de trabalho na unidade de Saúde, permitindo que os profissionais médicos se concentrem no 'core' de suas atividades, que é prestar cuidados de Saúde com qualidade.

O padrão HL7 data de 1987 é hoje usado em larga escala em hospitais e outros ambientes de Saúde ao redor do mundo. No Brasil, apesar do incentivo e da crescente adoção, a implantação integral do padrão em unidades de Saúde, tal como hospitais, ainda é considerada uma tarefa relativamente complexa. Existem diferenças significativas entre implementações do HL7 dentro de unidades hospitalares no que diz respeita à completude da norma.

Com vistas a ampliar a adoção da filosofia proposta pelo padrão HL7, em 2011 foi desenvolvido o padrão HL7 FHIR - *Fast Interoperable Health Resources*. O FHIR é o protocolo de comunicação mais avançado do conjunto de padrões HL7. Ele foi proposto a fim de aprimorar os padrões de troca de mensagens adotados pelo HL7 (BENDER; SARTIPI, 2013) (D'ANGELO; FERRETTI; GHINI, 2017). Ele foi projetado para ser mais fácil de implementar e mais extensível do que as versões 2.x e V3 do HL7. O padrão FHIR descreve novos formatos e elementos de dados específicos de Saúde, como um novo protocolo de troca eletrônica de informações de saúde utilizando APIs, além de usar terminologias já reconhecidas internacionalmente (WALINJKAR, 2018). Em particular, o HL7 FHIR é projetado para a *Internet*, aproveitando um conjunto de padrões *Web* e tecnologias de APIs, como o protocolo RESTful, além de HTML para a integração da interface de usuário e JSON ou XML para representação dos dados e verificação.

A grande vantagem do HL7 FHIR é que ele oferece melhorias quando comparado a padrões existentes, nomeadamente os padrões HL7 V3 e CDA, nos seguintes aspectos: (i) as especificações são concisas e de fácil compreensão; (ii) existe uma diversidade de bibliotecas que as implementam; e (iii) a sua construção é baseada em padrões *Web* bem estabelecidos, tais como XMLI, JSON II, HTTP III e REST IV. Dessa maneira, o HL7 FHIR é adequado para uso em uma ampla variedade de contextos, aplicativos móveis, comunicação em nuvem, compartilhamento de dados a base de RES, entre outros.

Algumas destas vantagens do HL7 FHIR sobre os formatos existentes incluem:

- Múltiplas bibliotecas de implementação;
- Desenvolvimento evolucionário do caminho do HL7 Versão 2 e CDA; com isso, os dois padrões podem coexistir;
- Fundação em padrões da *Web* - XML, JSON, HTTP, HTML, OAuth etc;
- Suporta a arquitetura RESTful;

- Possui um formato facilmente legível, o que facilita a leitura por parte dos usuários;
- Foco na implementação rápida, ou seja, fácil de entender para quem desenvolve, além de dispor de uma variedade de ferramentas disponíveis.

Em vista de tais características, o HL7 FHIR torna-se um facilitador na adoção de um padrão de comunicação em Saúde. Os esforços do HL7 FHIR visam, em última instância, simplificar e acelerar a adoção do padrão HL7 por ser facilmente consumível, mais robusto e por utilizar padrões abertos da *Internet* sempre que possível. Por usar formatos que são facilmente consumíveis pela *Web* o padrão evita a necessidade de ferramentas personalizadas e complexas. A documentação do FHIR disponibiliza exemplos para todas as implementações de artefatos e referência para várias plataformas, incluindo servidores de teste disponíveis.

A estrutura do HL7 FHIR possui mais de 100 diferentes componentes para conceitos do mundo real que cobrem a área da Saúde por inteiro, por exemplo, Paciente, Histórico Familiar, Médico, Dispositivo, Organização, Localização, Serviços de Saúde, incluindo, entre outros, Alergia, Plano de Assistência, Observação, Diagnóstico, Medicação etc. (SHOUMIK et al., 2018). Outro benefício importante do HL7 FHIR é que se trata de um padrão de código aberto que está disponível ao público por meio da *Internet*. A Figura 2 apresenta um exemplo de componente Paciente.

Enfim, o HL7 FHIR pode ser visto como um padrão que descreve duas coisas extremamente importantes para o compartilhamento eletrônico de registros de Saúde entre sistemas. A primeira é a estrutura de formato de dados para armazenar todos os tipos de registros médicos; e o segundo é a API (*Application Programming Interface*) sobre como acessar esses dados. O HL7 FHIR pode não resolver os desafios mais árduos da interoperabilidade na saúde, como políticas diferentes entre instituições, variabilidade nos dados capturados e o contexto em que estes são descobertos. No entanto, aponta como objetivo principal fazer da implementação da troca de dados o mais simples possível, para abrir caminho para a resolução das questões de interoperabilidade.

2.5 Formatos de troca de dados

Na Seção 2.4, apresentamos alguns dos principais padrões de informação em Saúde. Alguns destes padrões se apoiam em linguagens de marcação. As principais linguagens de marcação usadas pelos padrões adotados neste trabalho são introduzidos brevemente a seguir.

XML. O XML (*Extensible Markup Language*) é uma linguagem de marcação recomendada pelo W3C (*World Wide Web Consortium*) voltada para a criação e interpretação de conteúdos. O seu objetivo principal é facilitar o compartilhamento de informações atra-

Figura 2 – Exemplo do padrão HL7 FHIR

```
{
  "resourceType": "Observation",
  "id": "205",
  "source": "mitdb",
  "text": [
      "59 M 1957 694 x2",
      "Digoxin, Quinaglute",
      "The PVCs are of two forms, one of which is much more common than the other."
  ],
  "status": "final",
  "category": {
      "coding": {
          "system": "http://terminology.hl7.org/CodeSystem/observation-category",
          "code": "vital-signs",
          "display": "Vital Signs"
      },
      "text": "Vital Signs"
  },
  "code": {
      "coding": {
          "system": "http://loinc.org",
          "code": "85353-1",
          "display": "Vital signs, weight, height, head circumference, oxygen saturation and BMI panel"
      },
      "text": "Vital signs Panel"
  },
  "component": {
      "valueQuantity": 650000,
      "code": {
          "coding": {
              "system": "urn:oid:2.16.840.1.113883.6.24",
              "code": "131329",
              "display": "MDC_ECG_ELEC_POTL_V1"
          }
      },
      "valueSampledData": {
          "frequency": 360,
          "unit": "mV",
          "dimensions": 1,
          "data": [
              -0.54,
              -0.54,
              -0.54,
          ]
       }
   }
}
```


Fonte: Elaborado pelo autor

vés da *Internet* (GSMA, 2016). A linguagem é capaz de descrever diversos tipos de dados, independentemente de domínio. No domínio da Saúde, por exemplo, pode ser usada na especificação de prontuários médicos, na descrição de dispositivos de saúde, bem de outros dados relacionados ao domínio. XML é uma linguagem extensível, permitindo que novos dados sejam definidos através da utilização das marcas ou tags (< >), e tem a vantagem de permitir a descrição sintática de informações de forma legível tanto por humanos ou por sistemas de computador, com total independência de fornecedor de *hardware* e *software* (PAGLIARI et al., 2005; Ricardo Neubauer Moralles; ALEGRE, 2018). Além disso, XML também permite que os desenvolvedores dediquem de forma mais precisa seus esforços em outras tarefas do processo de desenvolvimento de *software*, deixando a tarefa de validação das estruturas de dados a cargo da especificação padrão da linguagem. Por fim, o XML é um padrão definido e recomendado pela W3C para a anotação de documentos. Apesar

da sua flexibilidade, robustez e relativa simplicidade, o XML apresenta alguns pontos fracos em relação a aplicações que demandam de maior complexidade, tais como vetores, listas associativas (chave valor) e informações relativas a configuração. No cenário de monitoramento remoto, algumas situações demandam associar um conjunto de chave e valor, por exemplo, a data e hora sendo a chave e o registro o valor. Além disso, no decorrer do dia esse procedimento pode ser realizado mais de uma vez, formando assim uma lista de valores. Alguns desses problemas foram estudados e solucionados por outros formatos, tais como o JSON e o YAML. Desta forma, apesar da simplicidade proposta pelo XML, em determinados cenários ele pode ser substituído por outros formatos mais simples para representação dos dados.

JSON. O *JavaScript Object Notation* (JSON) é um formato leve de troca de e dados que tem como principais características a sua simplicidade e universalidade, apresentando fácil leitura e escrita por humanos e máquinas. JSON é composto por um conjunto de chave e valor. As chaves representam os nomes dos atributos da classe e os valores o conteúdo do objeto (ALHOMSI et al., 2018). Embora o JSON seja um formato baseado em texto e independente da linguagem, ele é um subconjunto da linguagem de programação JavaScript. Como tal, possui uma série de benefícios como alta velocidade na execução e transporte de dados, por proporcionar arquivos com tamanho reduzido. Algumas das principais diferenças entre os modelos JSON e XML incluem: o JSON não é uma linguagem de marcação, não possuindo *tags* (< >) de abertura e de fechamento, com a representação das informações ocorrendo de forma muito compacta, quando comparada ao XML. O formato JSON é tipicamente usado em ambientes onde o tamanho do fluxo e o formato dos dados entre o cliente e servidor é extremamente importante, como no cenário de Saúde. Dessa forma, para sistemas mais modernos, o JSON é mais utilizado para comunicação do que o XML (KARAGIANNIS et al., 2016).

YAML. Segundo (BEN-KIKI; EVANS; INGERSON, 2009), o YAML *Ain't Markup Language* (YAML) é um formato de serialização e uma linguagem descritiva de dados semelhante ao XML, porém com a gramática muito mais compreensível. O formato apresenta características de boa legibilidade, boa interação com outras linguagens, boa capacidade expressiva e expansibilidade, além de facilidade de uso por ser altamente interpretável por humanos. Embora não seja menos genérico que o XML e JSON, o YAML é em grande parte mais simples de ler, editar, modificar e produzir que o XML. Ou seja, quase tudo o que é possível de representar em XML pode ser representado em YAML e, ao mesmo tempo, de uma forma mais compacta. Por possuir uma sintaxe limpa, torna a serialização de dados mais legível à inspeção humana (leitura e edição), além de ser muito bem estruturada. Sem perder a generalidade, o YAML apresenta maneiras alternativas de expressar o mesmo tipo de dados. Deste modo, no cenário de monitoramento remoto onde o tamanho do fluxo de dados é uma preocupação e aumenta com demanda, o YAML pode ser boa solução para a serialização de dados, tendo em vista que os arquivos configurados

nesse formato oferecem sintaxe simples e podem armazenar uma considerável quantidade de dados.

2.6 Simulação de aplicações de saúde

Em computação, um simulador é um programa que representa o comportamento de um sistema ao longo do tempo. Em alguns casos, o sistema simulado é real e já opera; porém, em outros, o sistema ainda não foi implementado ou mesmo projetado (COUTINHO et al., 2018). As motivações por trás do uso da simulação são muitas, várias delas vinculadas ao custo. Além dos custos, existe uma outra razão fortemente relacionada ao domínio de aplicação em Saúde: a realização de testes em sistemas reais pode ser às vezes perigosa - e eventualmente ocasionar perda de vidas - ou mesmo inviável ou impossível de serem realizados por razões diversas. Além disso, muitas e diferentes configurações podem ter de ser consideradas para dar suporte ao design do sistema, encarecendo ainda mais os custos de testes usando sistemas reais.

O uso de simuladores para validar aplicações e dispositivos de Saúde aumentou acentuadamente, em parte devido à maior conscientização da importância da segurança do usuário final mas também devido à evolução das tecnologias empregadas (OJIE; PEREIRA, 2018). Atualmente, simuladores são aplicados à pesquisa e desenvolvimento de novas ferramentas para área de Saúde, algumas dessas utilizadas para o monitoramento e tratamento de doenças em pacientes crônicos. Os simuladores permitem que os desenvolvedores controlem o ambiente de testes, padronizem as avaliações, validem os componentes em larga escala com as ferramentas apropriadas e, ainda, protejam os pacientes contra quaisquer tipos de erros gerados durante a avaliação (BHOJWANI, 2007). Portanto, a simulação no ambiente de Saúde visa fornecer um conjunto realista e econômico de ferramentas para melhorar a segurança das aplicações. Por outro lado, a simulação permite que os seus usuários revisem e pratiquem os testes quantas vezes forem necessárias para alcançar a proficiência, sem danificar ou prejudicar nenhuma parte envolvida.

Simular o aumento do número de pacientes monitorados e o consequente aumento do volume de dados na aplicação é um ponto muito importante no cenário de monitoramento remoto. Neste sentido, o uso de tecnologias que promovam a escalabilidade, por exemplo, a virtualização de sistemas, bem como a necessidade de simular os mecanismos de comunicação empregados na conexão com os domicílios dos pacientes, isto é, os protocolos de comunicação com os dispositivos de saúde, têm papel fundamental no ambiente de testes. Tais questões são tratadas nas seções seguintes.

2.7 Virtualização de sistemas com Docker

Como visto, reproduzir o comportamento de um sistema de Saúde de grande porte não é uma tarefa nada trivial. Por exemplo, adquirir ou alugar recursos de *Data Center* em larga escala que reflita com precisão o aumento e as configurações dinâmicas dos recursos disponíveis, como visto em cenários de Saúde, pode ser inviável tanto operacional quanto financeiramente (LO et al., 2012). Além disso, a necessidade frequente de mudanças nas configurações dos testes em larga escala envolve muita configuração manual, tanto do ambiente de *hardware* quanto a parte de *software* do ambiente de teste, tornando a avaliação muito demorada e custosa (PONTES; LIMA; FARIA, 2018). Como resultado, a reprodução do ambiente real se torna extremamente difícil. Por vezes, também é impraticável configurar um cenário realístico de teste com centenas ou milhares de nós para testar a escalabilidade do sistema manualmente, devido às inúmeras execuções de testes em diferentes condições e variedade de dispositivos de diversos fabricantes, modelos, sistemas operacionais e protocolos de comunicação.

Diferentes ferramentas de virtualização têm desempenhado um papel de destaque no provisionamento de ambientes para execução de simulação de aplicações complexas, como as de Saúde. O Docker é uma implementação de virtualização baseada em contêineres que vem conquistando cada vez mais espaço na comunidade de engenharia de *software* e sistemas distribuídos devido a todas as suas funcionalidades e facilidades oferecidas (Docker Inc., 2019). O Docker é um projeto de código aberto que se baseia em muitas tecnologias familiares de pesquisa de sistemas, tais como a virtualização de sistemas operacionais (VASE, 2015).

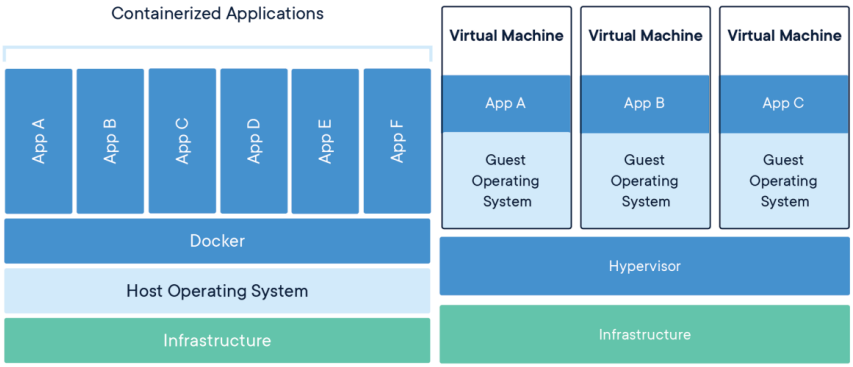
O Docker pode ser definido como uma plataforma de *software* que permite a criação de contêineres. Um contêiner, por sua vez, pode ser descrito como um pacote leve, autônomo e ainda executável, que inclui tudo o que é necessário para uma aplicação ser executada, por exemplo, código executável, ambientes de tempo de execução, ferramentas do sistema, configurações e bibliotecas do sistema (BOETTIGER, 2015). De forma mais simples, um Docker contêiner consiste em um sistema operacional, os arquivos adicionados pelo usuário, e os metadados.

Um contêiner é, portanto, um objeto de *software* que encapsula o aplicativo e suas dependências, permitindo que ele seja executado de forma isolada das outras aplicações em um mesmo computador. As aplicações que são executadas em um contêiner Docker e utilizam a virtualização em nível de sistema operacional no Linux são equivalentes às instâncias tradicionais de uma VM (*Virtual Machine*). No entanto, como apresentado na Figura 3, ao contrário das imagens de VM, os contêineres não agrupam um sistema operacional completo (HOSSAIN et al., 2017). Uma das grandes vantagens com o uso do Docker é a criação de cópias instantâneas de determinada funcionalidade em uma imagem, simplificando a implantação da mesma em outros contêineres e permitindo, assim,

a replicação de estados de contêineres em escala (BOETTIGER, 2015).

O uso da abordagem por contêineres tem se popularizado entre os desenvolvedores pela sua flexibilidade, portabilidade e escalabilidade. Mesmo as aplicações complexas podem ser conteinerizadas e transmitidas na *Internet* em um arquivo de tamanho menor em comparação com as imagens de VM. Assim, é possível criar uma imagem de contêiner localmente, armazená-la na nuvem ou distribuir e executar automaticamente réplicas na rede.

Figura 3 – Diferença entre máquina virtual e contêiner



Fonte: Adaptada de (Docker Inc., 2017)

A abordagem do Docker e a utilização da tecnologia de contêineres tem o potencial de melhorar o processo de desenvolvimento, simulação e de provisionamento de aplicações de Saúde. Dentre essas possíveis melhorias, destaca-se a simplificação da construção de ambientes de teste para as equipes de desenvolvimento e a redução de falhas decorrentes da diferença de configuração entre ambientes de qualidade, homologação e produção. Assim, é possível diminuir o tempo de elaboração dos testes de sistemas e favorecer os experimentos de dimensionamento pois o Docker fornece mecanismos que possibilitam aos desenvolvedores testarem seus aplicativos com funções de gerenciamento de recursos, virtualização e orquestração de serviços em ambientes realísticos com alterações mínimas.

2.8 Protocolos de comunicação

Esta seção introduz alguns protocolos usados na comunicação com dispositivos IoT usados em ambientes de monitoramento remoto de pacientes. Tais protocolos, usados na proposta apresentada nesta dissertação, são brevemente introduzidos a seguir.

2.8.1 MQTT (Message Queue Telemetry Transport)

O MQTT é um protocolo de comunicação desenvolvido por Andy Stanford-Clark, da IBM, e Arlen Nipper, da *Arcom Control Systems Ltd* (Eurotech) (NAIK, 2017). O MQTT é um protocolo da camada de aplicação projetado para comunicações M2M (*Machine-to-Machine*) em redes de dispositivos com restrições de recursos, como os dispositivos IoT empregados no cenário de monitoramento remoto de pacientes descrito neste trabalho.

O protocolo segue a arquitetura cliente/servidor com um mecanismo de troca de mensagens no estilo *publish/subscribe*. Todo dispositivo é um cliente que se conecta a um servidor, denominado de *broker*. Os clientes podem se inscrever em vários tópicos e receber todas as mensagens publicadas para cada tópico. Em outras palavras, o MQTT é orientado a mensagens, sendo que todas as mensagens devem ser publicadas para um endereço, chamado de "tópico"(NAIK, 2017). Os clientes, por sua vez, podem se inscrever em mais de um tópico, recebendo todas as mensagens enviadas para tais tópicos. O MQTT é um protocolo binário e normalmente requer cabeçalho fixo de 2 *bytes* com pequenas cargas úteis de mensagens até o tamanho máximo de 256 MB. Usa-se comunicação TCP para este fim.

A confiabilidade das mensagens no MQTT é atendida por três níveis de Qualidade de Serviço (*Quality of Service* - QoS). Cada mensagem deve ter seu QoS previamente selecionado ou devem manter o QoS 0 como padrão. Os níveis de QoS do MQTT são:

- QoS 0: no máximo uma vez. As mensagens são enviadas utilizando-se a capacidade total do sistema ambiente. Podem ocorrer perdas de mensagens.
- QoS 1: ao menos uma vez. As mensagens são asseguradas de chegar ao destino. Entretanto, podem ocorrer mensagens duplicadas.
- QoS 2: exatamente uma vez. As mensagens são asseguradas de chegar ao destino em uma única vez.

Sendo um protocolo da camada de aplicação, o MQTT depende dos protocolos TCP e IP nas camadas subjacentes. Entretanto, quando comparado ao HTTP, o MQTT possui uma sobrecarga menor de protocolo, sendo assim mais adequado para grandes redes de pequenos dispositivos que precisam ser monitorados ou controlados a partir de um servidor back-end na *Internet*, como é o cenário de aplicação desenvolvido nesta dissertação (DINCULEANă; CHENG, 2019).

2.8.2 CoAP (Constrained Application Protocol)

O protocolo CoAP é um protocolo *Web* que se diferencia do MQTT por rodar sobre o protocolo UDP (*User Datagram Protocol*). Ambos, porém, são capazes de operar em

ambientes restritos e são projetados principalmente para a "Internet das Coisas". O CoAP é uma variante do protocolo síncrono mais usado da *Web*, o HTTP (*Hypertext Transfer Protocol)*, tendo sido desenvolvido especificamente para suportar comunicações M2M em aplicações IoT. O protocolo segue o estilo arquitetural REST (*Representational State Transfer*), o qual consiste em princípios, regras e *constraints* que, quando seguidas, permitem a criação de um projeto com interfaces bem definidas e que fornecem interoperabilidade entre sistemas, ou seja, REST fornece um jeito mais simples do modelo servidor-cliente (MAZZER; FRIGIERI; PARREIRA, 2015).

Diferentemente do MQTT o CoAP usa o *Universal Resource Identifier* (URI) em vez de tópicos (NAIK, 2017). Dados são publicados por um editor no URI e o assinante assina recursos específicos indicados pelo URI. Além disso, quando novos dados são publicados no URI, todos os assinantes são notificados sobre o novo valor. O CoAP é um protocolo binário e normalmente requer cabeçalho fixo de 4 *bytes* com pequenas cargas úteis de mensagens até o tamanho máximo, dependendo do servidor da *Web* ou da tecnologia de programação.

O CoAP oferece mais funcionalidade do que o MQTT, pois suporta a negociação de conteúdo para expressar uma representação preferida de um recurso, ou seja, permite que cliente e servidor evoluam independentemente, adicionando novas representações sem afetar um ao outro (NAIK, 2017). Em parte, é similar ao HTTP, porém como os dispositivos de IoT normalmente operam em ambientes limitados, os pacotes do CoAP são muito menores. Para economizar espaço, os pacotes são passados no espaço de memória de um pacote anterior (DASTJERDI; BUYYA, 2016).

Por utilizar UDP o protocolo CoAP agrega simplicidade no funcionamento e economiza em processamentos. No entanto, como o UDP não é inerentemente confiável, o CoAP fornece seu próprio mecanismo de confiabilidade. Outra diferença entre CoAP e o MQTT é a disponibilidade de diferentes níveis de QoS. O MQTT define três níveis de QoS, enquanto o CoAP não fornece qualidade de serviço diferenciada para o ambiente de monitoramento.

2.8.3 WebSocket

O protocolo *WebSocket* é uma tecnologia que permite a comunicação bidirecional (*full-duplex*) por canais que podem transmitir e receber dados sobre um único *socket* TCP. O protocolo foi projetado para ser executado em navegadores (*browsers*) e servidores *Web* que suportam o HTML5, mas pode ser utilizado em qualquer cliente ou servidor de aplicativos (COMMUNITY, 2015). A especificação *WebSocket* define uma API que estabelece conexões *socket* entre um navegador da *Web* e um servidor, ou seja, *WebSockets* estabelecem uma conexão persistente entre cliente e o servidor e ambas as partes podem começar a enviar dados a qualquer momento.

O protocolo opera em duas partes: um mecanismo de *handshaking* e a transferência de dados propriamente dita. O protocolo *WebSocket* foi construído no topo do protocolo TCP e depende deste para iniciar a comunicação, que acontece após a execução do mecanismo de *handshaking* do TCP, firmando que as conexões estão prontas para iniciar a comunicação (DIZDAREVIĆ et al., 2019). A partir do estabelecimento da conexão, o servidor está ciente de todas as conexões *WebSocket* e pode se comunicar com cada uma individualmente. O mecanismo de *handshaking* garante que o servidor esteja sincronizado com seus clientes.

A API do *WebSocket* fornece um método para criar conexões seguras, o que é benéfico para fins de autenticação no cenário de Saúde. Utilizando o *WebSocket*, o servidor pode enviar dados diretamente para o cliente sem a pesquisa constante do cliente para novas atualizações. O *WebSocket* permanece aberto para o servidor ou o usuário enviar mensagens a qualquer momento até que um deles feche a sessão.

Em suma, o protocolo *WebSocket* permite a interação entre um navegador *Web* (ou outro aplicativo cliente) e um servidor da *Web* com sobrecarga menor do que as alternativas como HTTP, facilitando a transferência de dados em tempo real de e para o servidor. Tendo em vista que os dados utilizados pelo simulador proposto são inicialmente carregados dos repositórios formando um arquivo com muitos registros, o que pode tornar o arquivo muito pesado para a manipulação utilizando os protocolos mais leves e com limitações de memória conhecidos, como o CoAP, MQTT e o HTTP, torna-se necessário o uso de um protocolo mais robusto para trafegar os pacotes de registros dentro do simulador para o ambiente simulado ou até mesmo dentro da aplicação.

2.8.4 Programação orientada a fluxo

A Programação Orientada a Fluxo, também referenciada na literatura como Programação Baseada em Fluxo, é um paradigma de programação proposto por J. Paul Rodker Morrison no final dos anos 60, que define aplicativos como redes de processos "caixa preta"que trocam dados através de conexões predefinidas por transmissão de mensagens, onde as conexões são especificadas externamente aos processos. Estes processos "caixa preta"podem ser reconectados infinitamente para formar aplicativos diferentes sem precisar serem alterados internamente (MORRISON, 2010).

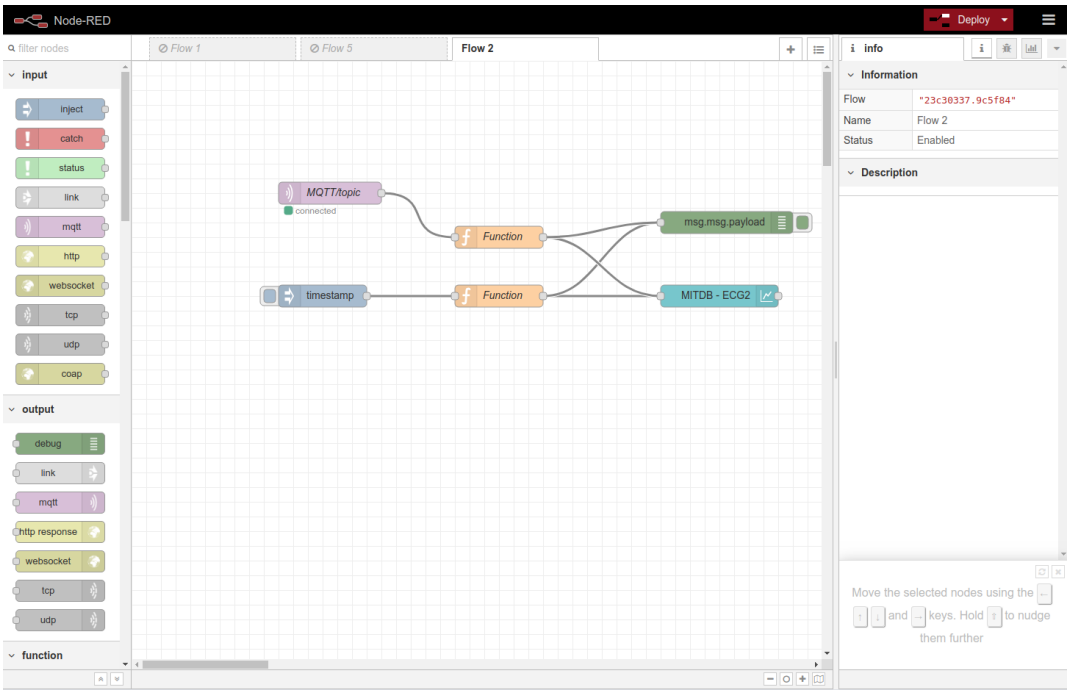
O Node-Red é uma ferramenta de programação visual que utiliza a programação orientada a fluxo de dados, de código aberto, e fundamentada na *Web*, adequada para a criação de aplicações IoT. Embora tenha sido desenvolvida para ser utilizada em "Internet das Coisas"pode ser utilizada para qualquer outro fim, por exemplo, para escutar eventos HTTP, *WebSocket*, *twitter*, TCP em aplicações orientadas a eventos.

O Node-Red foi criado e é mantido pela IBM. Ele foi implementado na linguagem

JavaScript utilizando a estrutura Nodejs, aproveitando, assim, o modelo de evento interno do Node e o suporte nativo ao JavaScript no editor do cliente e no servidor (BLACKSTOCK; LEA, 2014). De forma geral, o Node-Red permite que um usuário não-técnico crie aplicações de forma mais simples, por meio de uma interface gráfica de arrastar e soltar, que conecta diferentes componentes formando um "fluxo" de execução.

Os programas no Node-RED formam um conjunto de fluxos e estes fluxos são constituídos por nós conectados por linhas. A interface do usuário consiste em um editor de fluxo com modelos de nós que podem ser arrastados e soltos em uma tela de fluxo (Figura 4). O Node-RED é uma ferramenta flexível e muito adequada para a prototipação de aplicações. O sistema permite a criação rápida de aplicativos, especialmente aplicativos que são acionados a partir de eventos, o que o torna uma ferramenta interessante para o desenvolvimento de aplicações de Saúde baseadas em dispositivos IoT. A essência da ferramenta é permitir que usuários não especialistas simplesmente criem e configurem aplicativos em tempo real nos dispositivos finais (LEKIĆ; GARDAŠEVIĆ, 2018).

Figura 4 – Interface do Node-Red



Fonte: Elaborado pelo autor

A *interface Web* do Node-Red disponibiliza nós padrões, nós customizados por outros usuários, e também permite que o usuário desenvolva e personalize os seus próprios nós conforme a sua necessidade, a fim de executar funções específicas, tais como enviar dados dos sensores para serviços como o "Twitter"ou até mesmo executar análises complexas com facilidade.

O Node-RED foi usado para construir a aplicação desenvolvida no estudo de caso (prova de conceito) apresentado neste trabalho. Tal aplicação tem por objetivo demonstrar funcionalidades e potencialidades da ferramenta de simulação proposta neste trabalho.

2.9 Considerações finais do capítulo

Este capítulo apresentou uma visão geral do cenário de monitoramento remoto de pacientes em domicílio definido para este trabalho, bem como as tecnologias associadas ao cenário que foram empregadas na implementação do simulador proposto e da prova de conceito. A caracterização das etapas tradicionais e a identificação de elementos de um sistema de monitoramento de pacientes em domicílio permitiram observar os desafios tecnológicos que devem ser superados para a implantação de um ambiente de monitoramento em larga escala baseado em dispositivos de IoT.

O capítulo a seguir realiza um levantamento de ferramentas de simulação e outros trabalhos relacionados ao ambiente proposto nesta dissertação. O objetivo é identificar tecnologias e estratégias utilizadas na simulação de ambientes de Saúde e outros requisitos que devem ser atendidos na criação da plataforma de simulação de aplicações de Saúde proposta neste trabalho.

3 Trabalhos Correlatos

Este capítulo apresenta alguns trabalhos relacionados ao tema de pesquisa investigado nesta dissertação, com ênfase em trabalhos que apresentam ferramentas de simulação de dados biomédicos de pacientes no contexto de ambientes de atenção à Saúde. Dentre os vários trabalhos analisados na revisão bibliográfica, alguns foram selecionados por estarem mais diretamente relacionados aos objetivos propostos nesta dissertação. A avaliação dos trabalhos focou nas suas características funcionais e estratégias utilizadas para testar e validar aplicações de eHealth baseadas em IoT.

3.1 Ferramentas de Simulação em Ambientes de eHealth

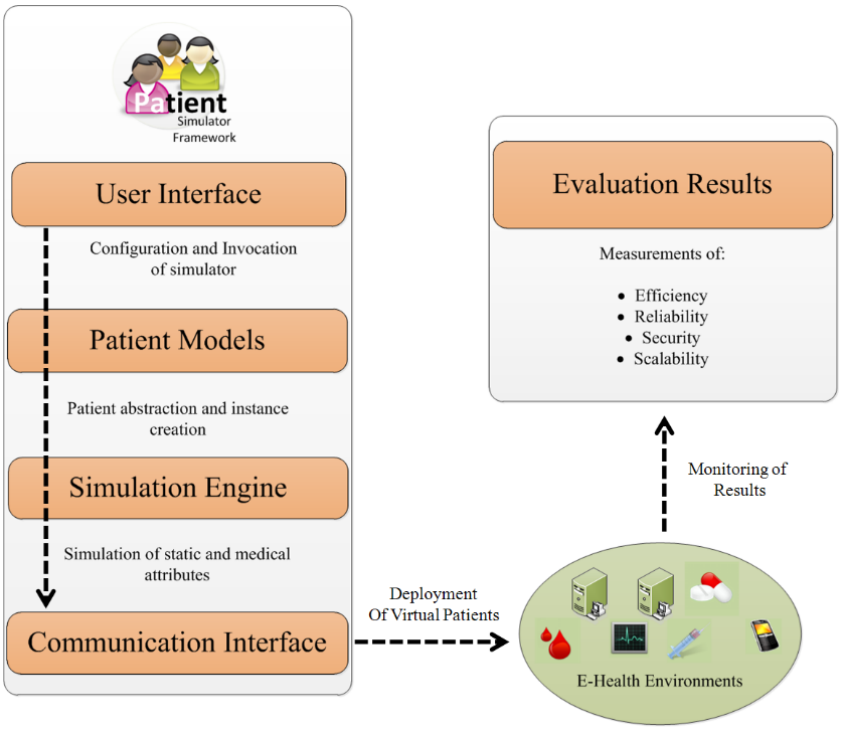
Os trabalhos apresentados em (LO et al., 2011), (AGAR; EREN; CINAR, 2006), (LOOGA; OU; DENG, 2012), (KERTESZ; PFLANZNER; GYIMOTHY, 2018), (PONTES; LIMA; FARIA, 2019), descrevem sistemas que simulam ou emulam dados fisiológicos de pacientes, os quais são consumidos por aplicações de eHealth. Uma breve descrição das características destes sistemas é apresentada nas próximas seções. Uma discussão comparativa sobre estes trabalhos à luz do cenário apresentado no Capítulo 2 é mostrada ao final do capítulo.

3.1.1 Patient Simulator: Towards Testing and Validation of eHealth Infrastructures

O trabalho de (LO et al., 2011) discute alguns dos grandes desafios em desenvolver infraestruturas e ferramentas de suporte eHealth que necessitam previamente da realização de testes e avaliações que apenas podem ocorrer após a aprovação legal e ética. O sistema Patient Simulator é apresentado com o objetivo de simular dados do paciente para a avaliar infraestruturas de eHealth e serviços em relação à eficiência, confiabilidade, segurança e escalabilidade em um estágio inicial da aplicação, sem a necessidade de utilizar dados reais do paciente. O simulador Patient Simulator é composto por quatro componentes, apresentados na Figura 5: Interface do Usuário, Modelos de Pacientes, Máquina de Simulação e Interface de Comunicação.

A Interface do Usuário proporciona automação e promove a simplicidade na realização das tarefas de teste e validação. Em outras palavras, este componente é responsável por simplificar a execução da aplicação. O segundo componente, Modelos de Pacientes, é responsável por tratar os modelos abstratos de pacientes. Ele faz isso separando os pacientes em duas categorias a partir do uso de um conjunto de atributos, por exem-

Figura 5 – Patient Simulator Framework



Fonte: Adaptada de (MODELING; LO, 2011)

plo, atributos médicos (nível de glicose, pressão arterial, alergias, etc.). É no terceiro componente onde o processo de simulação de atributos médicos e não médicos de um paciente ocorre. O mecanismo de simulação propõe o uso da simulação baseada em eventos discretos, combinado com o comportamento probabilístico e determinístico. Por último, a Interface de Comunicação é responsável por estabelecer o protocolo de comunicação com as infraestruturas eHealth, de acordo o método de comunicação suportado pela aplicação em teste e mediante a utilização dos seguintes padrões: XML, HTTP e WML (MODELING; LO, 2011).

Na pesquisa apresentada por (LO et al., 2011), a atual implementação do Patient Simulator limita a cinco a quantidade e os tipos de sinais fisiológicos tratados: frequência cardíaca, pressão arterial, temperatura corporal, níveis de oxigênio e frequência respiratória. O Patient Simulator utiliza valores gerados aleatoriamente para simular sinais vitais, ou seja, ele faz uso de técnicas de distribuição aleatórias. Embora essa técnica seja comum nas fases iniciais dos testes, ela não permite a simulação de cenários um pouco mais complexos, por exemplo, ataque cardíaco e derrame.

O Patient Simulator propõe-se a avaliar ambientes de eHealth, sob as métricas de eficiência, confiabilidade, segurança e escalabilidade. Embora o simulador desenvolvido não utilize métodos realísticos o suficiente para testar determinadas soluções de Saúde em

Figura 6 – Patient Simulator – interface gráfica do usuário

Fonte: Adaptada de (LO et al., 2011)

ambiente de desenvolvimento, o Patient Simulator apresenta vários aspectos importantes para o sucesso e uso deste tipo de sistema. Por exemplo, o projeto da sua interface gráfica de interação do usuário, como visto na Figura 6, é bem interessante e pode ser usado como referência na implantação de simuladores tais como o proposto nesta dissertação. Além disso, devido a modularização da arquitetura, as alterações realizadas em um componente do simulador não afetam a base de código dos outros componentes, característica importante a ser adotada em um simulador para área de Saúde, pois pode absorver com maior facilidade eventuais mudanças no cenário da Saúde.

3.1.2 Glucosim: Educational Software for Virtual Experiments with Patients with Type 1 Diabetes

O GLUCOSIM é um simulador de sinais fisiológicos apresentado por (AGAR; EREN; CINAR, 2006). O trabalho concentra-se na simulação de níveis de glicose e insulina no sangue de pacientes saudáveis e pacientes com diabetes tipo 1 para fins educacionais e de treinamento. Um banco de dados nutricional foi integrado ao GLUCOSIM para determinar a quantidade de carboidratos de uma refeição específica e atribuir o registro como uma entrada para o GLUCOSIM. O simulador pode ser usado para testar o desempenho de diferentes algoritmos de controle para bombas automatizadas para infusão de insulina.

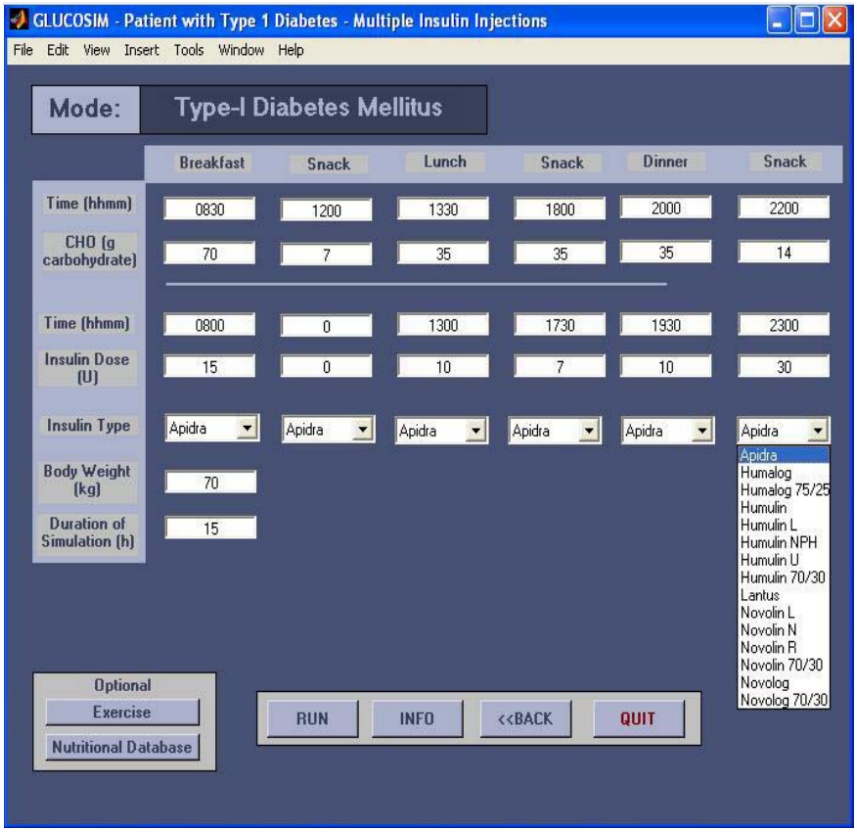
Por outro lado, o GLUCOSIM tem como objetivo auxiliar os usuários, em sua maioria estudantes de engenharia biomédica e química, na visualização das variações dinâmicas e concentração de glicose no sangue em resposta a variações do organismo do paciente e a fatores externos, como o consumo de alimentos e administração de insulina (ERZEN; BIROL; ÇINAR, 2001). Deste modo, o simulador propõe-se a demonstrar os efeitos de mudanças na ingestão de alimentos e terapia sobre os níveis de glicose em diabetes mellitus dependente de insulina.

O GLUCOSIM realiza a simulação da dinâmica dos níveis de glicose e insulina no sangue em pacientes diabéticos tipo I e em pessoas saudáveis fornecendo, assim, boas informações para os desenvolvedores de bombas de insulina e estudantes. Experimentos virtuais podem ser realizados com o GLUCOSIM, observando rápidas variações na concentração de glicose, em conformidade com as condições de dieta e exercício. Resultados como concentração de glicose no sangue e concentração de insulina no sangue ao longo do tempo podem ser salvos, podendo ser visualizados mediante o uso de ferramentas adequadas. O fluxo de decisões no uso de GLUCOSIM oferece a oportunidade de aprender sobre diabetes com vários recursos, por meio da seleção de diferentes parâmetros, por exemplo, quantidade e tipo de alimentos, peso corporal, exercício, tempo de injeção de insulina, volume da dose de insulina, tipo de insulina, injeção por seringa ou bomba, e bomba manual ou controlada automaticamente podem ser investigados em simulações com pacientes diabéticos.

O GLUCOSIM permite ainda a inserção de vários parâmetros importantes para a simulação do nível de glicose no sangue, tais como o tempo de refeição, ingestão de carboidratos, dosagem de insulina, peso do paciente. A partir da entrada desses dados os módulos do GLUCOSIM ficam responsáveis pela simulação do nível de glicose em pacientes com diabetes tipo 1, a geração de insulina no pâncreas de indivíduos saudáveis e por testar a tolerância à glicose por via oral. Uma limitação da pesquisa desenvolvida por (AGAR; EREN; CINAR, 2006) é que ela se concentra na simulação deste conjunto específico de sinais fisiológicos do paciente, não sendo possível simular outros dados biomédicos disponíveis em repositórios de dados de Saúde, que podem eventualmente ser de interesse da aplicação de eHealth que está sendo simulada.

Um dos pontos a serem destacados no trabalho é a simplicidade na interação do usuário com o simulador, em que menus, botões e controles deslizantes são utilizados como elementos de controle durante a simulação, como mostrado na Figura 7. Os resultados da simulação são exibidos graficamente. Os valores numéricos e gráficos podem ser salvos e baixados pelos usuários. A principal novidade em seu trabalho é a versatilidade empregada no simulador, permitindo a inserção de parâmetros, incluindo horário das refeições, ingestão de carboidratos, dosagem de insulina, peso do paciente entre outros parâmetros. Uma das técnicas adotadas pelo autor (AGAR; EREN; CINAR, 2006) ocorre

Figura 7 – Glucosim – interface gráfica do simulador



Fonte: Adaptada de (AGAR; EREN; CINAR, 2006)

durante a execução da simulação, em que o GLUCOSIM permite observar a correlação entre os parâmetros de entrada e a saída das concentrações de glicose e insulina por meio da interface disponibilizada.

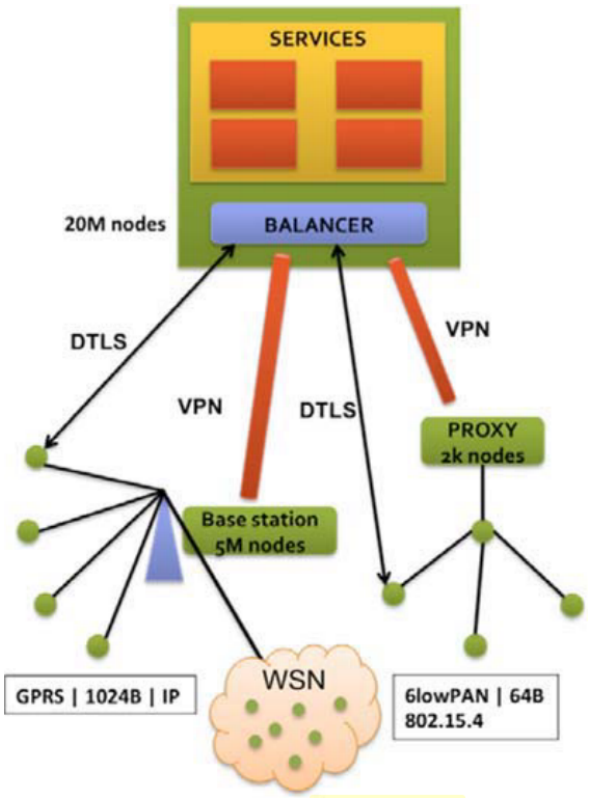
3.1.3 MAMMotH: a massive-scale emulation platform for internet of things

O trabalho de (LOOGA; OU; DENG, 2012) realiza uma revisão de literatura na área de simuladores e emuladores de larga escala. Através deste levantamento bibliográfico, os autores observaram que as ferramentas de teste pesquisadas falham em promover adequadamente a identificação e análise dos erros ocorridos nos aplicativos testados ou ainda não focam na visualização do comportamento em tempo real do programa analisado. O levantamento demonstrou também a carência de ferramentas de testes apropriadas para a compreensão do funcionamento de aplicações em redes de larga escala. Como resultado do levantamento, os autores propõem o MAMMotH, um emulador de IoT em larga escala, que pode ser usado para criar cenários de experimentos, implantá-los em uma plataforma de teste e monitorar os resultados.

Segundo os autores, o volume de registros no cenário de monitoramento remoto

de pacientes cresce exponencialmente à medida que se atualizam os equipamentos ou novos dispositivos de Saúde surgem. Deste modo, o ambiente pelo MAMMotH aborda um dos principais desafios no desenvolvimento de sistema de eHealth: a escalabilidade da solução (PONTES; LIMA; FARIA, 2019). Sendo assim, um dos requisitos fundamentais do MAMMotH é emular o tráfego de rede de diversos nós e torná-los o mais próximo possível da realidade. A plataforma de emulação planejada no escopo do projeto MAMMotH é incomum em sua escala e pode ser utilizada para criar cenários experimentais, implantá-los em uma plataforma de experimentação e monitorar os resultados. Para emular os nós, eles utilizam uma ferramenta baseado em Java e a libCoAP, que suporta o CoAP (*Constrained Application Protocol*), um protocolo de código aberto baseado no UDP (*User Datagram Protocol*), que permite maior escalabilidade e desempenho. Além disso, o emulador pode ser utilizado para reproduzir os problemas de comunicação presentes em um ambiente real ao qual os dispositivos estão conectados, simulando links de rádio para cada dispositivo, controlando, por exemplo, os atrasos e as perdas de mensagens.

Figura 8 – Visão geral do emulador MAMMOTH



Fonte: Adaptada de (AGAR; EREN; CINAR, 2006)

O MAMMotH emula milhares de dispositivos por meio do uso de Máquinas Virtuais (VM). A arquitetura do MAMMotH é apresentada na Figura 8 e ela pressupõe três cenários distintos. O primeiro cenário da arquitetura assume que dispositivos móveis serão

conectados via GPRS (*General Packet Radio Services*) a uma estação base que forma uma topologia estrela. No segundo cenário proposto um sistema independente de rede de sensores (WSN) é conectado a uma estação base via GPRS. Por último, a arquitetura apresenta o cenário em que dispositivos restritos (sensores) conectam-se a (*proxies*) que agem como um intermediário para as requisições dos clientes, interligando a parte *back-end* da aplicação (COUTO, 2013). Por sua vez, para reproduzir os problemas de comunicação presentes em um ambiente real, como os vistos no cenário de Saúde, o proxy ao qual os dispositivos estão conectados simula um link de rádio para cada nó, capaz de atrasar e encaminhar as mensagens. Por essa razão, os desenvolvedores de aplicações de Saúde podem utilizar esse tipo de configuração para simular o ambiente em que a aplicação eHealth funcionará considerando, por exemplo, a localização dos aparelhos de monitoramento em um ambiente com obstáculos que ocasionam perda de sinal e possíveis erros na ferramenta de monitoramento.

Com os cenários considerados e a escala massiva de dispositivos e pacientes no ambiente de saúde conectado às aplicações, o autor acredita que o MAMMotH é apropriado para emular a comunicação de um ambiente real ao qual os dispositivos estão conectados, tratando problemas comumente encontrados no cotidiano de uma aplicação eHealth, tais como perda de conexão, atraso e perda de dados durante o envio ou recebimento. Tais características tornam a validação e verificação da aplicação muito mais realística.

Um ponto que vale a pena observar no trabalho é tipo de topologia utilizada nos três cenários. A topologia estrela se parece muito com a configuração usada no ambiente de atenção à Saúde no domicílio, no qual existe um dispositivo *gateway* responsável por conectar os dispositivos a sua volta. Neste tipo de simulação, com a topologia bem definida, o processo de adição de novos dispositivos é simples, o gerenciamento é centralizado e falha de um dispositivo não afeta o restante da aplicação, permitindo assim a experimentação em escala da ferramenta. Além disso, o uso do protocolo CoAP adotado pelos autores do MAMMotH contribui para o aumento da disponibilidade dos serviços necessários para a realização de testes no ambiente de Saúde, visto que os pacotes do CoAP são muito menores em relação a outros protocolos de comunicação.

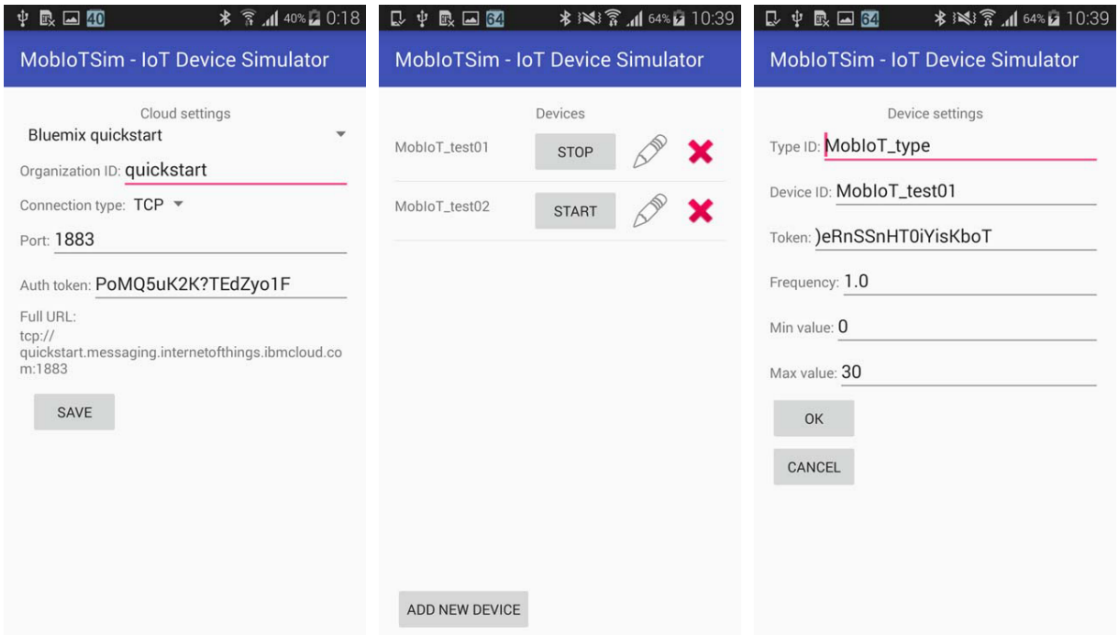
3.1.4 MobIoTSim: Towards a mobile IoT device simulator

O MobIoTSim é um simulador de dispositivos IoT que foi desenvolvido para ser executado na plataforma *Android* com a finalidade de auxiliar pesquisadores no desenvolvimento de aplicações IoT e, consequentemente, tornar a necessidade de sensores reais dispensáveis durante a fase inicial dos testes (PFLANZNER et al., 2016). O MobIoTSim pode ser utilizado para simular qualquer dispositivo IoT e suporta conexão com a nuvem para transmissão e recepção de dados. Além disso, o simulador ainda dispõe de um gerador de dados de sensores para um ou mais dispositivos. Deste modo, o MobIoTSim simula

dispositivos IoT e os valores críticos lidos dos sensores.

O MobIoTSim gera dados de sensores de forma aleatória em um intervalo de tempo determinado, fornecido como entrada no aplicativo. Além disso, o usuário pode especificar a frequência de envio de dados para todos os dispositivos. As principais etapas são representadas por três partes do aplicativo, apresentadas na Figura 9: a tela de configuração de nuvem, os dispositivos e as configurações dos dispositivos. O simulador utiliza o protocolo de comunicação MQTT para enviar e receber dados, mediante a utilização da biblioteca Eclipse Paho. Os dados que transitam pelo aplicativo são representados por um objeto JSON estruturado e compatível com o formato de mensagem do IBM IoT Foundation (PFLANZNER et al., 2016). O MQTT utilizado pelo MobIoTSim para troca de dados é leve em relação aos requisitos de processamento e, como tal, presta-se bem a aplicativos em redes com limitação de largura de banda e restrições de latência, como aquelas normalmente encontrados em ambientes de Saúde.

Figura 9 – Capturas de tela do aplicativo MobIoTSim Android: Configurações de nuvem, Dispositivos, Configurações do dispositivo



Fonte: Adaptada de (KERTESZ; PFLANZNER; GYIMOTHY, 2018)

Os autores do MobIoTSim (PFLANZNER; FIDRICH; KERTESZ, 2017) também desenvolveram um serviço de *gateway* privado nas plataformas IBM Bluemix e Microsoft Azure que podem se conectar ao simulador para gerenciar dispositivos e enviar notificações ao aplicativo quando houver algum valor crítico do sensor. Deste modo, pesquisadores e desenvolvedores podem empregar essa ferramenta para examinar o comportamento dos sistemas de IoT durante o desenvolvimento e para avaliar aplicativos IoT com mais

eficiência. A solução do *gateway* pode ser personalizada e disponibilizada em um contêiner Docker, tornando assim a solução mais portátil e escalável.

O objetivo principal do trabalho é fornecer uma plataforma para que pesquisadores compreendam o manuseio do dispositivo IoT sem que necessitem comprar sensores e dispositivos extras (KERTESZ; PFLANZNER; GYIMOTHY, 2018). Sendo assim, por meio do uso do MobIoTSim os usuários podem explorar o funcionamento detalhado dos sistemas de IoT e ajudar na criação de um entendimento inicial de um sistema completo de IoT integrado. Em conformidade com o objetivo desta dissertação, o simulador proposto possui aspectos importantes para a execução de testes no cenário de saúde, como a capacidade de simular diversos dispositivos IoT e o serviço de *gateway* em nuvem personalizável com a capacidade de gerenciar muitos dispositivos simulados (PFLANZNER et al., 2016). Além disso, a interface do MobIoTSim proporciona flexibilidade para que o usuário possa criar um dispositivo, configurar as simulações e também permite o acompanhamento do experimento.

Embora o método utilizado para a troca de dados entre os dispositivos e a nuvem seja simplificado, há algumas limitações no trabalho. Além do MobIoTSim estar limitado ao protocolo de comunicação MQTT para a comunicação entre o ambiente IoT e a plataforma em nuvem utilizada para visualização, o simulador é dependente da plataforma privada Bluemix ou Azuree para conectar dispositivos e a nuvem para processar dados, reagir a eles ou realizar visualizações durante a execução dos testes. Na versão gratuita do Bluemix, por exemplo, o número de dispositivos é limitado a vinte dispositivos, 1 GB de espaço de armazenamento e o envio de 100 MB de dados para os dispositivos no período de 30 dias.

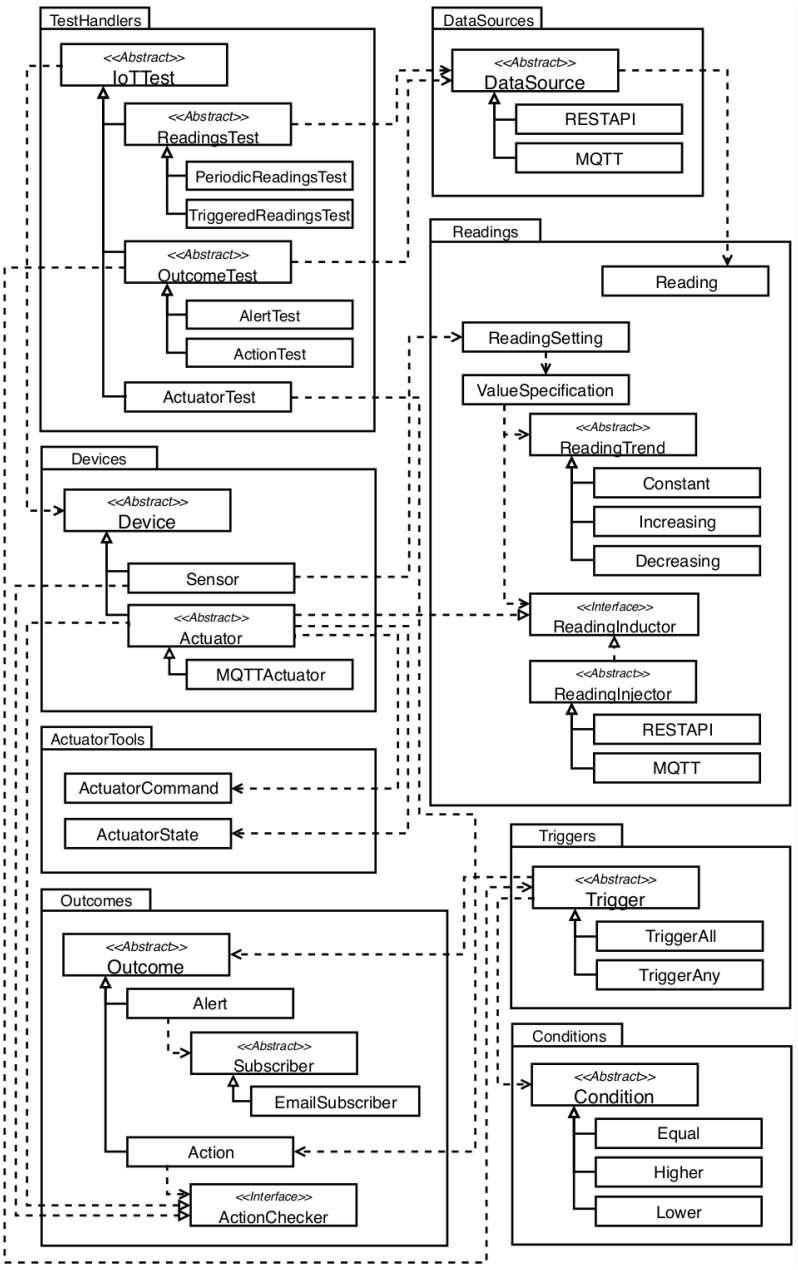
3.1.5 Izinto: a pattern-based IoT testing framework

Izinto é uma estrutura de teste de IoT baseada em padrões que visa apoiar o processo de teste de integração de ecossistemas de IoT (PONTES; LIMA; FARIA, 2019). O Izinto torna possível a execução de testes de comportamentos recorrentes no escopo de IoT de maneira automatizada e sem a necessidade de preocupação com a lógica do teste, tendo em vista que o *framework* implementa de maneira genérica um conjunto de padrões de teste específicos e que podem ser instanciados para cenários de IoT. Deste modo, os padrões propostos pelo *framework* visam reduzir o esforço alocado na etapa de configuração dos testes das aplicações IoT.

O *framework* Izinto implementa os seguintes padrões de teste e suas funcionalidades: (i) testes de leituras periódicas, responsáveis por verificar se um sensor é capaz de realizar leituras a uma taxa fixa e se essas leituras estão corretamente transmitidas e persistidas no sistema IoT; (ii) testes de leituras acionadas, encarregadas de verificar se um sensor é capaz de realizar medições acionadas por um usuário e se essas leituras estão corretamente transmitidas e, ainda, se estão persistidas no sistema; (iii) testes de alerta, responsáveis por

verificar se os alertas são enviados para assinantes pré-definidos sempre que uma condição é atendida; (iv) testes de ação, que verificam se as ações definidas são executadas sempre que uma condição é atendida; e, por último, (v) teste de atuadores, que verificam se um atuador é capaz de executar comandos, ou seja, se é capaz de alterar seu estado conforme comandado (PONTES; LIMA; FARIA, 2018).

Figura 10 – Arquitetura do Framework Izinto



Fonte: Adaptada de (DE; DO; FOR, 2018)

Na arquitetura apresentada pela Figura 10, é possível distinguir dois módulos principais: um que engloba a lógica de teste real (pacote TestHandlers) e outro correspondente

às abstrações dos componentes de IoT (outros pacotes). A lógica de teste foi implementada usando JUnit um *framework* de código aberto e que segue padrões já citados anteriormente. O segundo módulo é formado por um conjunto de conectores para os componentes reais da IoT – sensores, atuadores e fontes de dados, que permitem configurar e controlar os referidos componentes em companhia de um conjunto de classes que representam conceitos como leituras e comandos do atuador, com alertas e ações (DE; DO; FOR, 2018).

O Izinto foi validado dentro de um cenário AAL onde vários sensores coletam e transmitem dados dos parâmetros de saúde do paciente quanto às condições ambientais. Os alertas são gerados quando a condição do paciente é determinada para exigir atenção e ações pré-definidas são acionadas por alterações nas condições ambientais. Izinto visa dois perfis diferentes. Por um lado, ele pode ser usado por especialistas em testes para testar praticamente qualquer solução de IoT, já que a estrutura pode ser estendida para atender a uma necessidade específica. Por outro, aqueles com o mínimo conhecimento técnico sobre testes ainda podem utilizar a estrutura, pois apenas algum conhecimento sobre o sistema testado é necessário para configurá-la (PONTES; LIMA; FARIA, 2019).

Os padrões de testes identificados no trabalho concentram-se mais nos testes de recursos, relacionados a requisitos funcionais. O Izinto possui uma estrutura bem definida de arquitetura de testes que permite a identificação de erros introduzidos ou ocasionados por algum acidente no ambiente de Saúde, o que faz dele um trabalho inspirador para a proposta apresentada nesta dissertação.

3.2 Discussão

A seguir, é feita uma breve discussão dos trabalhos apresentados na seção anterior tendo por base as demandas do cenário de monitoramento de pacientes em domicílio descrito no Capítulo 2. Os seguintes critérios de comparação foram escolhidos: (I) Interface Humano-computador; (II) Escalabilidade; (III) Padrões de testes; (IV) Uso de repositórios de dados de Saúde; (V) Protocolos de comunicação.

(I) INTERFACE HUMANO-COMPUTADOR. Como sabido, o objetivo fundamental da Interface Humano–Computador é promover a usabilidade e a comunicabilidade entre os computadores e os humanos. Segundo (SULISTIO; YEO; BUYYA, 2004), uma interface humano-computador gráfica fornece um meio muito mais simplificado de executar simulações em comparação com uma interface orientada por linha de comando. Uma boa interface pode facilitar o uso, agilizar a realização de tarefas de teste e validação e tornar a visualização da execução da solução de Saúde muito mais simples. Em outros termos,na área de Saúde as interfaces humano-computador são essenciais na construção de sistemas usáveis, seguros e funcionais. O Patient Simulator, simulador de dados proposto por (LO et al., 2011), apresenta uma interface interativa, fácil de usar e de rápido aprendizado. A

interface proposta tem como vantagem a redução de erros durante a configuração, além de propiciar maior interação e entendimento do usuário em comparação com uma interface orientada por linha de comando; entretanto, como já mencionado, o sistema limita-se a simular exclusivamente cinco tipos de sinais biomédicos por meio da geração dinâmica de dados; em consequência, a estratégia utilizada pelos autores, não permite a reprodução de cenários um pouco mais complexos, como doenças crônicas e ataque cardíaco. Devido à limitação no número de sinais observados não é possível explorar a heterogeneidade de um número maior de dispositivos utilizados em sistemas de monitoramento durante as simulações, bem como executar testes de alerta e teste de atuadores. Por sua vez, o projeto GLUCOSIM, que propõe uma plataforma de simulação de sinais biomédicos de pacientes saudáveis e com diabetes do tipo 1, possui uma interface que permite configurar vários parâmetros de entrada, a saída de glicose e as concentrações de insulina no sangue e no fígado (AGAR; EREN; CINAR, 2006). No entanto, o GLUCOSIM concentra-se na simulação de apenas um sinal fisiológica específico, limitando também o seu uso. Já o MobIoTSim disponibiliza a interação entre o simulador e o usuário por meio de um aplicativo, que permite adicionar dispositivos e configurar o ambiente de teste. O usuário pode também acompanhar e enviar notificações durante as simulações por meio de um *gateway* configurado na plataforma Bluemix ou Azure. O Izinto hospeda um painel através do qual as leituras coletadas pelo *gateway* podem ser visualizadas em tempo real, também permite a visualização de dados históricos e alertas.

(II) ESCALABILIDADE. Como visto anteriormente, a escalabilidade tem a ver com a capacidade de um sistema, rede ou processo de se adaptar para lidar com uma quantidade crescente de trabalho (MODELING; LO, 2011). Dada a grande quantidade e variedade de dispositivos que podem estar envolvidos no cenário de monitoramento domiciliar e o potencial aumento da população alvo deste serviço, os sistemas e aplicativos de monitoramento precisam ser construídas atendendo aos requisitos de escalabilidade a fim de suportar o crescente número de usuários e de dispositivos IoT no ambiente domiciliar. O sistema MAMMotH (LOOGA; OU; DENG, 2012) possui uma arquitetura para emulação de dispositivos em larga escala capaz de lidar com o crescimento de diferentes cenários por meio de uma interface Web. O MAMMotH utiliza o CoAP, protocolo que possui pacotes de mensagens menores do que outros protocolos de comunicação e uma máquina virtual que contém um proxy simulando um link de para cada nó. Esta estratégia de uso de um protocolo leve junto às VMs visa facilitar a promoção da escalabilidade do emulador. O sistema MobIoTSim, por sua vez, aborda a questão de escalabilidade mediante a utilização da tecnologia Docker contêiner. Como já mencionado, esta tecnologia fornece uma camada adicional de abstração com o uso da virtualização de sistemas e tal abordagem é usada para criar instâncias de *gateways*, simulando o aumento do número de dispositivos e usuários que os utilizam nos seus domicílios. Por sua vez, o GLUCOSIM (AGAR; EREN; CINAR, 2006) não trata de escalabilidade, pois seu objetivo é mais voltado para auxiliar

na visualização das variações dinâmicas e concentração de insulina.

(III) PADRÕES DE TESTES. As infraestruturas de eHealth nem sempre retratam as mesmas visões que os profissionais de saúde têm sobre os requisitos necessários para as soluções eHealth. A natureza heterogênea e distribuída dos sistemas de IoT e de Saúde dificultam os testes de soluções. Por outro lado, essas aplicações evidenciam a importância dos testes, dando origem à necessidade de uma solução eficiente e eficaz para implementar testes automatizados pois erros neste cenário podem ter sérias consequências. Apesar disso, há uma falta de processos e métodos de teste padrão, o que representa um grande desafio para os testes de IoT. Embora algumas soluções já estejam disponíveis para fins de teste de ferramentas de saúde, é possível identificar várias limitações, como as relacionadas à falta de suporte para diferentes tecnologias de habilitação de IoT, a falta de funcionalidades prontas para uso e a impossibilidade de melhoria por meio de extensão. Deste modo, o Izinto (PONTES; LIMA; FARIA, 2019) define cinco padrões de teste que podem ser instanciados para cenários de IoT e eHealth. Levando em consideração as particularidades desses sistemas, é possível identificar um conjunto de estratégias de teste para analisar comportamentos recorrentes que podem ser descritos por padrões de teste. A abordagem baseada em padrões de testes utilizada pelo Izinto promove a reutilização tanto no ambiente de ferramentas IoT quanto no cenário de Saúde, pois os padrões de teste apresentados podem ser aplicados a vários cenários para testar comportamentos recorrentes.

(IV) REPOSITÓRIOS DE DADOS DE SAÚDE. Como também já mencionado, o armazenamento, a preservação e o acesso aos dados de pesquisa são elementos fundamentais para o ciclo de vida da pesquisa e para a expansão do conhecimento. Sabe-se que o compartilhamento de dados de pesquisa é uma questão crucial para a disseminação e otimização da ciência e, neste sentido, os repositórios de dados de pesquisa são considerados uma fonte de dados valiosa e confiável para tal objetivo. No caso da área de Saúde, os repositórios de dados simplificam o acesso aos dados biomédicos, que são produzidos e colocados à disposição dos profissionais e desenvolvedores de soluções para tornar possível não apenas a sua leitura e acompanhamento, mas também a sua reutilização em novos projetos na área, por exemplo, na validação de novas aplicações de monitoramento remoto de pacientes.

Os repositórios de dados biomédicos online e gratuitos têm papel de destaque nos meios científicos, clínicos e educacionais, por terem a importante função social de permitirem, acesso aberto a uma variedade de dados de sinais fisiológicos reais, além de outros importantes dados relacionados à Saúde para uso da comunidade da área médica, pesquisadores e desenvolvedores de aplicações de Saúde. Uma outra vantagem apresentada por estes repositórios é que o conjunto de dados são bem descritos, tendo em vista que passam pela revisão cuidadosa de muitos usuários, o que permite uma rápida descoberta de erros o que, por sua vez, gera mais confiança e precisão no processo de simulação. O uso

de um conjunto de dados já conhecido reduz ainda a responsabilidade do desenvolvedor de aplicações em demonstrar que seus resultados são baseadas em bons dados e métodos.

Existe uma ampla variedade de repositórios de dados de Saúde online, no entanto, um que se destaca é o PhysioNet, repositório utilizado por (LO et al., 2011) para validar o simulador Patient Simulator e escolhido para ilustrar esta funcionalidade na plataforma de simulação proposta nesta dissertação. O PhysioNet é um repositório visto como uma comunidade de pesquisa online, oferecendo gravações de sinais fisiológicos multiparâmetros e *software* de processamento, simulação e análise de sinais de código aberto (CIUCU; METHODS, 2015). O PhysioNet apoia o acesso aos dados para diversas finalidades, tais como pesquisa básica, clínica, fisiologia, saúde pública e engenharia biomédica.

Em que pese a sua importância, o uso de repositórios abertos de dados biomédicos ainda é pouco frequente nos simuladores voltados para a área de Saúde, particularmente nos simuladores IoT. Entre tantos desafios, um deles é escolher um padrão adequado para a troca eletrônica de dados com vistas à interoperabilidade, pois cada repositório varia em termos de conteúdo, objetivos, métodos e políticas de acesso.

(V) PROTOCOLOS DE COMUNICAÇÃO. Protocolos de Comunicação possibilitam a conexão e transferência de dados entre dois ou mais sistemas computacionais. Tecnicamente, um protocolo pode ser definido como um conjunto de normas que caracterizam o formato, a sincronização, a sequência e também a detecção de erros e falhas na comutação de pacotes que duas aplicações ou dispositivos devem seguir para que a comunicação entre eles permaneça estável e funcional. É assim que qualquer aplicação consegue enviar e receber mensagens instantâneas dentro do contexto de aplicações inteligentes. O Izinto suporta as tecnologias mais amplamente utilizadas em relação a dispositivos e fontes de dados, sendo assim suporta dispositivos que se comunicam via MQTT e a API REST. Além disso, o Izinto pode ser estendido à medida que outros padrões de teste surgem ou como especialistas em testes encontram a necessidade de oferecer suporte a outras tecnologias devido a sua estrutura modular (DE; DO; FOR, 2018). O Patient Simulator propõe uma estrutura de comunicação com aplicações eHealth utilizando protocolos padronizados (MODELING; LO, 2011), como exemplo, XML, HTTP e WML. A escolha do protocolo utilizado dependerá inteiramente do método de comunicação suportado pela aplicação em teste. Com o objetivo de definir suporte às implementações futuras o simulador define a interface de comunicação como um componente separado para que as futuras implementações do Patient Simulator possam ser facilmente atualizadas e estendidas sem outros componentes. O MAMMotH busca emular o tráfego de rede de milhões de nós para torná-los o mais próximo possível da realidade por meio de nós que se conectam à rede via links de rádio, como GPRS ou 802.15.4. Assim, os autores procuram simplificar a comunicação da aplicação mediante a simulação das características de links de rádio e empregando o protocolo CoAP que visa simplificar o uso de redes. Por sua vez,

o MobIoTSim (PFLANZNER et al., 2016) pode simular um ou mais dispositivos IoT e é implementado como um aplicativo móvel para a plataforma *Android*. O simulador é capaz de gerenciar dispositivos utilizando os protocolos de IoT mais populares como o MQTT e HTTP combinados com diferentes estruturas de formatos de dados. A geração de dados do sensor dos dispositivos simulados são valores gerados aleatoriamente na faixa fornecida pelo usuário e enviados de acordo com a frequência já definida utilizando a biblioteca Eclipse Paho.

Tabela 1 – Comparação das características dos trabalhos correlatos

Simulador/Emulador	(I)	(II)	(III)	(IV)	(V)
(LO et al., 2011)	Sim	Sim*	Não	Não	Sim
(AGAR; EREN; CINAR, 2006)	Sim	Não	Não	Não	Não
(LOOGA; OU; DENG, 2012)	Sim	Sim	Não	Não	Sim
(PFLANZNER et al., 2016)	Sim	Sim	Não	Não	Sim
(PONTES; LIMA; FARIA, 2019)	Sim	Sim*	Sim	Não	Sim

Fonte: Elaborado pelo autor

A Tabela 1 apresenta o suporte que cada simulador oferece aos requisitos definidos pelos critérios de comparação (I) a (V) usados na avaliação dos trabalhos. As células marcadas com 'Não' representam requisitos que não foram identificados ou para os quais não havia informações suficientes no estudo. As células marcadas com 'Sim' representam requisitos observados e cumpridos pelo trabalho analisado. Nos casos em que o requisito foi parcialmente observado, a célula contém o valor 'Sim*'.

3.3 Considerações Finais do Capítulo

A análise dos trabalhos correlatos permitiu observar importantes características de diversas ferramentas de *software* desenvolvidas com o propósito de avaliar/simular sistemas inteligentes de Saúde, com diferentes abordagens e atendendo a múltiplos objetivos. Foi possível comprovar que, apesar dos méritos particulares destas ferramentas, pode-se identificar lacunas funcionais nas arquiteturas destes sistemas, por exemplo, a ausência de interfaces interativas centradas simultaneamente nas necessidades específicas do usuário da área de Saúde e de outros *stakeholders* do domínio, particularmente quanto à integração e visualização mais adequada dos dados do paciente; à limitação de funcionalidades determinada pelo propósito da ferramenta e cenário escolhido, que pode, eventualmente, restringir a simulação de cenários IoT mais complexos; à ausência de dados reais de

pacientes disponíveis em repositório dados públicos visando a validação da aplicação; a falta da possibilidade de escolha dos padrões de comunicação M2M (*Machine-to-Machine*) com os dispositivos IoT, dentre outros requisitos.

Evidencia-se, portanto, a carência de ferramentas de simulação que integre essas funcionalidades e facilitem a reprodução de dados de Saúde em escala de usuários, contemplando por exemplo, uma unidade de saúde, um bairro, ou até uma cidade, permitindo assim aos desenvolvedores de soluções IoT para a área de Saúde personalizar e escalar o ambiente de simulação de acordo com as suas reais necessidades e mais facilmente validar as aplicações de Saúde consumidoras destes dados.

4 O Simulador HealthSimulator

Este capítulo apresenta a arquitetura da plataforma HealthSimulator, um ambiente onde testes de aplicações IoT em Saúde podem ser conduzidos de maneira repetida e controlada. O projeto da arquitetura visou tratar, de forma integrada, algumas questões e desafios identificados nos capítulos anteriores. Inicialmente, são apresentados os principais requisitos a serem atendidos pelo simulador, seguido da proposta da sua arquitetura conceitual. O capítulo discute ainda as escolhas tecnológicas adotadas na implementação da plataforma.

4.1 Requisitos

Inspirado nos sistemas apresentados no Capítulo 3 e nas características e demandas do cenário escolhido, são elencados os seguintes requisitos a serem atendidos pelo simulador proposto:

1. **Deve permitir a execução e o monitoramento da simulação via interface gráfica.** O sistema deve permitir acesso às configurações e à visualização das informações resultantes das simulações por meio de uma interface gráfica amigável, simples e transparente, ao invés de linha de comando. Esta interface deve permitir a execução das simulações de forma objetiva, possibilitando a seleção dos principais parâmetros que regem o processo de simulação, incluindo a escolha dos protocolos de comunicação com os dispositivos IoT, o número de *gateways* e os tipos de sinais biomédicos a serem simulados.

2. **Deve adotar uma abordagem leve e ágil para tratar questões de escalabilidade.** A fim de suportar cenários com grande número de pacientes e dispositivos IoT conectados simultaneamente, a infraestrutura proposta deve ser capaz de permitir instanciar, de forma ágil e segura, novos 'nós' no ambiente simulado. Para tal, deve fazer uso de tecnologias de virtualização e empacotamento de aplicações, formando uma camada adicional de abstração que permita escalar vários usuários e dispositivos.

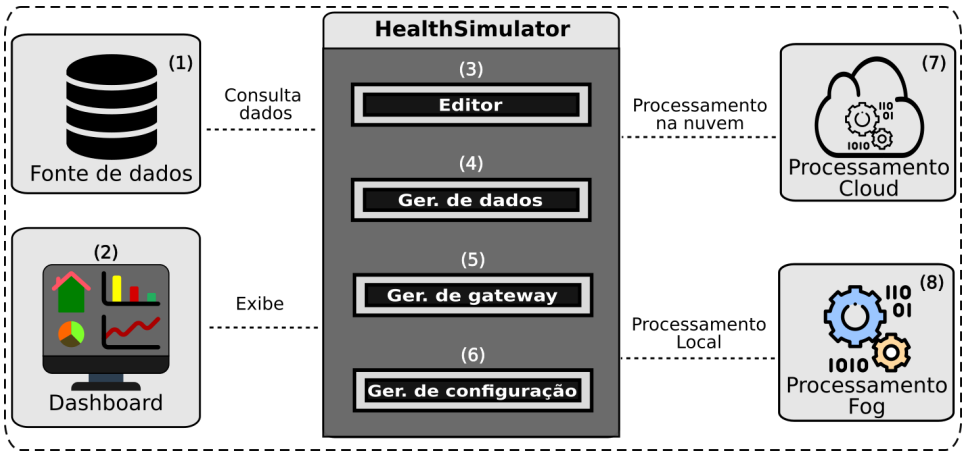
3. **Deve coletar sinais biomédicos de repositórios de dados de saúde**. Diferentemente da maioria das abordagens descritas no Capítulo 3, o sistema proposto deve facilitar o acesso e o uso de dados reais de pacientes nos experimentos de simulação, obtidos de repositórios online de Saúde. O uso de registros biomédicos reais permitirá a simulação de cenários diversos de observação de sinais fisiológicos de pacientes saudáveis e com doenças crônicas nas aplicações de monitoramento remoto.

4. **Deve ser capaz de se comunicar com as aplicações de monitoramento de saúde por meio de mecanismos de comunicação M2M em IoT.** O simulador deve interagir com as aplicações de sistema de saúde por meio de interfaces de comunicação leves, adequadas para o ambiente de IoT em Saúde, que atendam aos requisitos de processamento limitado e de baixos recursos dos dispositivos usados em cenários de monitoramento remoto.

4.2 Arquitetura Conceitual

A plataforma de simulação proposta é introduzida nesta seção. Sucintamente, do ponto de vista operacional, o sistema basicamente acessa dados de repositórios de Saúde disponíveis na sua interface de configuração, coleta as condições fisiológicos dos pacientes monitorados e as entrega a um gerenciador de dados. O gerenciador de dados é responsável por analisar o conteúdo recebido e convertê-lo para um formato padronizado, que possa ser utilizado pelas diversas aplicações de eHealth (GSMA, 2016). Estes dados padronizados são então encaminhados para um outro componente, o item (5) da Figura 11, responsável por escalar e carregar estes dados nas aplicações eHealth que serão validadas. As aplicações, por sua vez, processam os dados conforme a abordagem mais conveniente aos objetivos da simulação, realizando o processamento destes dados na nuvem ou mesmo numa camada "fog", mais próxima aos dispositivos de monitoramento.

Figura 11 – Visão geral da arquitetura conceitual do HealthSimulator



Fonte: Elaborado pelo autor

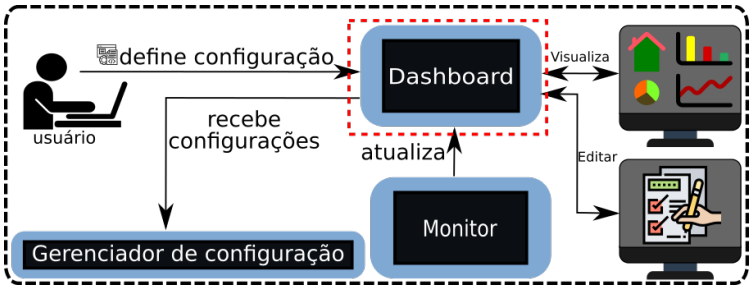
A arquitetura conceitual do HealthSimulator mostrada na Figura 11 é composta pelos seguintes módulos: (1) *Dashboard*, (2) Gerenciador de dados, (3) Gerenciador de gateway, (4) Gerenciador de configuração, e (5) Gerenciador de conectividade, os quais são

detalhados a seguir. As aplicações e aerviços eHealth (7 e 8), usuárias dos dados coletados pelo simulador, são também mostrados na figura.

4.2.1 Módulo: Dashboard

O módulo *Dashboard* compõe a Interface Humano-Computador proposta para a plataforma. Este módulo tem como funcionalidades principais a configuração e o monitoramento das simulações por meio de uma interface gráfica simples, que facilita a interação entre o usuário e a plataforma de simulação. A interface visual proposta permite que o usuário crie modelos de simulação de maneira simples e rápida, facilitando a definição das preferências ou contextos dos usuários. Assim, o usuário pode construir o seu modelo de simulação definindo poucas configurações, tais como o tipo de dado biomédico e o endereço do dado escolhido nos repositórios disponíveis.

Figura 12 – Dashboard – Interface de comunicação com o usuário



Fonte: Elaborado pelo autor

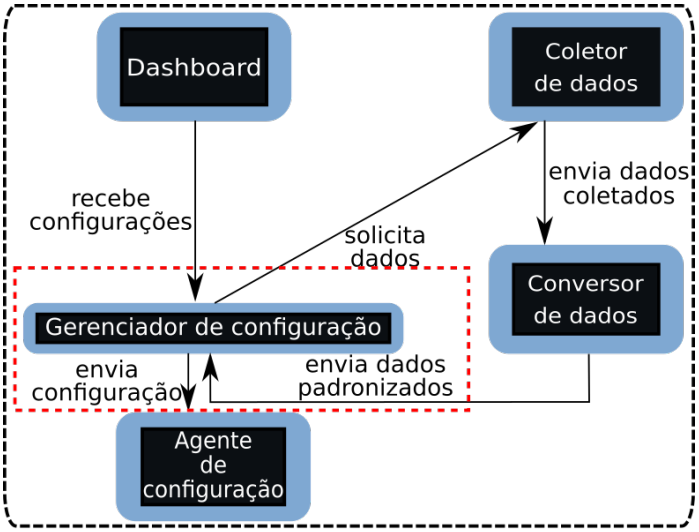
O usuário define as configurações por meio de um editor inserido no *Dashboard* que dá suporte às configurações de testes iniciais. Durante a configuração do *Dashboard*, é permitido ao usuário escolher dentre os principais formatos de serialização de dados utilizados por ferramentas eHealth: JSON, YAML ou XML (GSMA, 2016). Além disso, outros parâmetros, como a quantidade de usuários na aplicação de destino, o repositório de dados a ser acessado e o tipo de sinal biomédico monitorado pela aplicação a ser testada podem ser configuradas.

Como pode ser observado na Figura 12, o usuário primeiramente define os parâmetros da simulação no editor do `Dashboard`. Em seguida, os critérios de simulação são encaminhados para o módulo `Gerenciador de configuração`, que se comunica com o ambiente simulado para elencar outras etapas da simulação. Outro integrante da plataforma observado na figura é o `Monitor`, componente responsável por acompanhar alterações, parâmetros gerados pela simulação e por atualizar as informações na área de visualização do `Dashboard`.

4.2.2 Módulo: Gerenciador de dados

O módulo `Gerenciador de dados` é responsável por administrar o acesso aos repositórios *online* disponíveis, armazenar localmente os dados biomédicos para a simulação, bem como por definir os métodos que manipulam ou alteram os registros coletados, além de ser responsável por converter os registros em um formato acessível e comum. O `Gerenciador de dados`, mostrado na Figura 13, é formado por dois componentes principais: o `Coletor de dados` e o `Conversor de dados`. O `Coletor de dados` é encarregado de realizar as consultas aos dados requisitados pelo usuário nos repositórios disponíveis, bem como por armazená-los. Por sua vez, o `Conversor de dados` é responsável pelo processo de transferência dos dados brutos coletados dos repositórios para um formato que seja compreensível pelas aplicações eHealth.

Figura 13 – Gerenciador de dados



Fonte: Elaborado pelo autor

O `Coletor de dados` interage com repositórios distintos, formados por arquivos de diferentes tipos de sinais fisiológicos utilizados pela comunidade médica, que demandam formas de interação particulares. Assim, o Coletor de Dados é, na verdade, um elemento conceitual formado por um conjunto de componentes individuais de encapsulamento - "*wrappers*", cada um dos quais responsáveis por interagir com um particular repositório, por exemplo, PhysioNet, Kaggle etc. Os repositórios listados na plataforma HealthSimulator incluem coleções de sinais cardiopulmonares, eletrocardiograma (ECG), nível de glicose e outros dados biomédicos de indivíduos saudáveis e/ou pacientes com uma variedade de condições com importantes implicações para a Saúde. Essas coleções incluem dados de uma ampla gama de estudos, desenvolvidos com apoio de membros da comunidade de pesquisa.

Na interação com os repositórios, os registros de dados são obtidos pelo `Coletor de dados` de forma transparente e padronizada, por meio das respectivas bibliotecas disponíveis. Por exemplo, na interação com o PhysioNet, o HealthSimulator interage com dois dos seus componentes principais: (I) o PhysioBank, formado por um conjunto de registros de sinais biomédicos, e (II) o PhysioToolkit, um conjunto de ferramentas para manipulação dos dados.

O fluxo de operação do `Coletor de dados` se inicia quando ele recebe uma requisição disparada pelo `Gerenciador de configuração`. Em seguida, ele analisa o registro solicitado pelo usuário no repositório e, caso o registro seja válido, é feita uma cópia local do arquivo do conjunto de informações requeridas. Caso o registro especificado não seja encontrado no repositório, o usuário receberá uma notificação de que os registros não foram achados. O `Coletor de dados` também é encarregado por processar e organizar as requisições entrantes localmente, além de enviar para o `Conversor de dados` os registros biomédicos para a conversão.

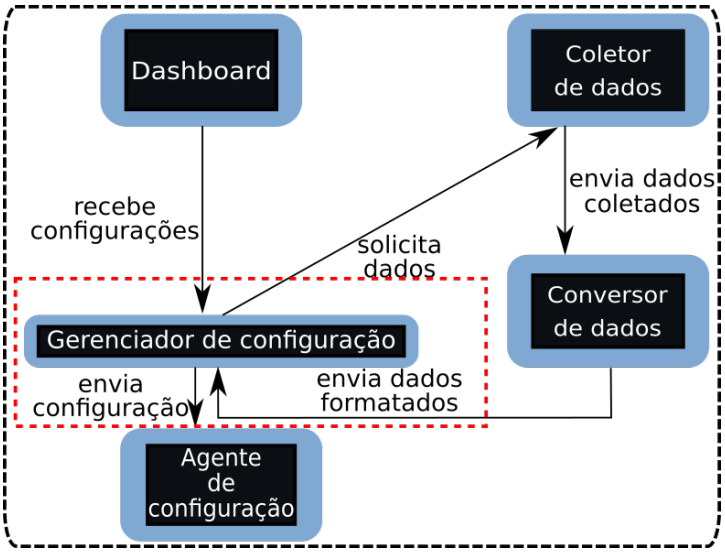
Os registros solicitados pelo usuário, válidos, são encaminhados e processados pelo `Conversor de dados` para um formato de dados comum no ambiente de Saúde, e útil para as aplicações. Durante a configuração, o usuário deve selecionar o campo com o formato de saída dos dados que deseja utilizar em sua aplicação, por exemplo, XML, CSV, JSON, YAML entre outros. Segundo (ABUKHOUSA; MOHAMED; AL-JAROODI, 2012), no âmbito da Saúde, ainda não há padrões bem estabelecidos e disponíveis para que os desenvolvedores de soluções eHealth possam utilizar durante o projeto e construção dos seus sistemas. Isso inclui definições de tipos de dados, frequência de captura, forma de armazenamento dos registros (compactação) e o tipo de privacidade. Muitos desses padrões são desenvolvidos por diferentes organizações e, às vezes, não são interoperáveis ou não são diretamente coordenados uns com os outros em nível organizacional (GSMA, 2016). Portanto, em um ambiente de simulação de aplicações de Saúde é muito importante que se ofereça suporte a diferentes formatos de dados para as aplicações consumidoras. Este procedimento de conversão de dados pode, portanto, economizar tempo e recursos valiosos durante o desenvolvimento de soluções.

Neste sentido, visando à interoperabilidade entre as aplicações, o HealthSimulator adota o padrão HL7 *Fast Healthcare Interoperability Resources* (FHIR) (GSMA, 2016) para a troca eletrônica de dados. O FHIR define um conjunto de "Recursos" que representam conceitos clínicos granulares que podem ser gerenciados isoladamente ou agregados em documentos complexos. Basicamente, o FHIR é projetado para a *Web* e os recursos são baseados em formatos de dados simples, por exemplo, CSV, YAML, XML ou JSON. Além disso, cada recurso possui um localizador uniforme e previsível de recursos (URL).

4.2.3 Módulo: Gerenciador de configuração

O componente `Gerenciador de configuração` é responsável por receber as configurações do usuário e por delegar funções para os demais componentes do simulador. Além disso, ele também se encarrega de estabelecer a comunicação da plataforma de simulação e o ambiente simulado por meio de mensagens. Sendo assim, o componente `Gerenciador de configuração` funciona como uma ponte entre o simulador e o ambiente simulado, sempre buscando garantir que as mudanças na configuração sejam mantidas. Provê, assim, a integridade do ambiente, minimizando a possibilidade de erros em itens de configuração.

Figura 14 – Gerenciador de configuração



Fonte: Elaborado pelo autor

O `Gerenciador de configuração` é um componente síncrono. Isto significa que mesmo após a execução inicial de alguma instrução e encaminhamento das configurações para o ambiente simulado ele permanece ativo, aguardando por novas configurações. Em outras palavras, este componente permanece em constante espera por novas interações entre o usuário e o simulador ou mesmo uma interação entre simulador e repositórios, seja ela de configuração ou retorno da transferência dos dados do repositório ao simulador. O primeiro tipo de interação, por exemplo, garante que de maneira dinâmica, sem a necessidade de paradas, o usuário realize variadas alterações na configuração do simulador, realizando mudanças no estado, número de repetições, alteração no formato de saída dos registros e tipo de registro biomédico, dentre outras. A Figura 14, apresenta as interações do componente `Gerenciador de configuração` com os outros elementos da arquitetura.

O componente `Gerenciador de configuração` transforma as configurações definidas pelo usuário em comandos e requisições para os demais elementos, o que favorece a interoperabilidade entre os elementos da arquitetura. A forma de inicialização deste

componente será melhor descrita na seção de implementação, porém, de forma sucinta, ele recebe e carrega um arquivo de configuração contendo todas as configurações do sistema no formato de dados JSON. A linguagem JSON foi escolhida por criar um meio simples de interação entre o usuário e o módulo do simulador, além de tornar o arquivo de configuração legível para humanos e facilmente interpretável pela máquina. O processo de carga da configuração é baseado em recursos de validação que o JSON traz, como o JSON *Schema* (LATVAKOSKI et al., 2014). Dessa forma, qualquer erro na maneira como o usuário configura o simulador pode ser detectado antes de sua execução, diminuindo a possibilidade de erros posteriores no seu funcionamento. Uma vez que a configuração é validada, o próximo passo é inicializar a simulação.

4.2.4 Módulo: Agente de configuração

O `Agente de configuração` representado na Figura 15, acompanhado do `Gerenciador de configuração`, são responsáveis por estabelecerem a comunicação entre as duas camadas da arquitetura. O `Agente de configuração` é encarregado de receber as configurações básicas de simulação definidas pelo usuário por meio do `Gerenciador de configuração`. Além disso, ele realiza a implantação das configurações no `Gerenciador de gateway`. Portanto, a cada vez que houver alguma alteração na configuração do simulador, ele realizará as mudanças necessárias na topologia.

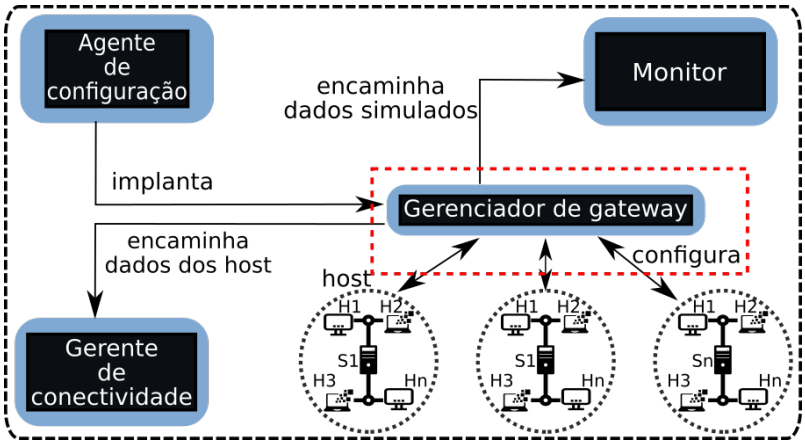
O `Agente de configuração` também é responsável por controlar os estados do simulador e por coordenar a execução das tarefas. O simulador opera por meio de quatro estados: (I) o estado inicial de preparação ("*prepare*"), em que as configurações são implantadas e os dados biomédicos do paciente são encaminhados para serem reproduzidos na aplicação avaliada; (II) o estado "*pause*", que é utilizado quando já se tem alguma simulação em andamento, isto é, quando o estado de preparação estiver atribuído a alguma simulação já existente e válida; (III) o estado "*resume*", responsável por retornar a simulação no mesmo estado em que ela foi pausada e, por fim, (IV) o *status* "*stop*", encarregado de parar a simulação e limpar os dados existentes.

Em síntese, o `Agente de configuração` recebe mensagens do `Gerenciador de configuração` com o "*status*"a ser operado pelo simulador, as configurações de saída dos dados de saúde (por exemplo, protocolo, formato, número de conexões e repetições) e o endereço dos registros do repositório selecionado. O agente desempacota as configurações, faz a requisição dos dados via requisição HTTP utilizando as configurações recebidas e inicia os outros módulos responsáveis pela simulação em si e pela comunicação com a aplicação sendo avaliada.

4.2.5 Módulo: Gerenciador de gateway

O módulo de `Gerenciador de gateway` é responsável por escalonar a execução dos registros biomédicos requeridos pelo usuário nas aplicações de saúde durante as simulações, permitindo, assim, configurar uma grande quantidade de nós de forma automatizada e reproduzir, portanto, o ambiente de Assistência à Autonomia no Domicílio. O módulo automatiza o processo de escalonamento das aplicações de Saúde por meio da instalação e configuração de aplicativos de monitoramento de pacientes, o que é realizado via criação de um ambiente isolado como gerenciador de contêiner (HOSSAIN et al., 2017). O gerenciador de contêineres agrupa partes de *software* de um sistema de arquivo completo, abrangendo todos os recursos necessários para a execução de um ambiente simulado. Dessa forma, o usuário não necessita lidar com as etapas de configuração manual de dezenas, centenas ou mesmo milhares de nós, simulando o ambiente AAL por completo para avaliar sua solução de forma escalonada.

Figura 15 – Gerenciador de gateway



Fonte: Elaborado pelo autor

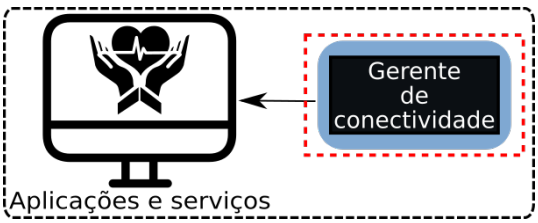
A Figura 15, apresenta as conexões do módulo `Gerenciador de gateway` com os outros módulos da arquitetura. À medida que o `Gerenciador de configurações` recebe as definições e os registros especificados pelo usuário, ele os envia para o `Agente de configuração`, que vai se incumbir de carregar as configurações no ambiente virtual, portáveis para qualquer outra aplicação. O `Agente de configuração` interage com o gerenciador de contêiner ajustando os parâmetros de configuração, por exemplo, a quantidade de *hosts virtuais*, *switches*, controladores e largura de rede. Basicamente, o `Agente de configuração` é incumbido da implantação das configurações no `Gerenciador de gateway`. Com as configurações do ambiente virtual devidamente definidas, o `Gerenciador de gateway` configura o *layout* da rede, o tráfego de informações e como os dispositivos serão conectados. Para isso, o gerenciador empacota uma aplicação ou o ambiente inteiro

dentro de uma imagem criando, assim, um contêiner com todas as configurações necessárias para executar a simulação, o que é representado na Figura 15 pelos *hosts*. O gerenciador de contêineres isola uma virtualização no nível de sistema operacional utilizando apenas o *Kernel* de seu hospedeiro e, com isso, permite que um ou mais contêineres trabalhem com recursos compartilhados, porém isolados. Isso promove a escalabilidade.

4.2.6 Módulo: Gerente de conectividade

O componente `Gerente de conectividade`, destacado na Figura 16, é o elemento do simulador responsável pela troca de mensagens entre o simulador e as aplicações de Saúde utilizando os protocolos determinados pelo usuário durante a etapa de configuração. Este componente implementa uma interface de conexão genérica que pode ser implementada de acordo com a necessidade de conexão da aplicação eHealth. Devido a generalidade existente na área de de Saúde, existe a necessidade que este componente da arquitetura possa ser atualizado ou receba uma nova implementação de acordo com o cenário de atuação.

Figura 16 – Gerente de conectividade da plataforma HealthSimulator



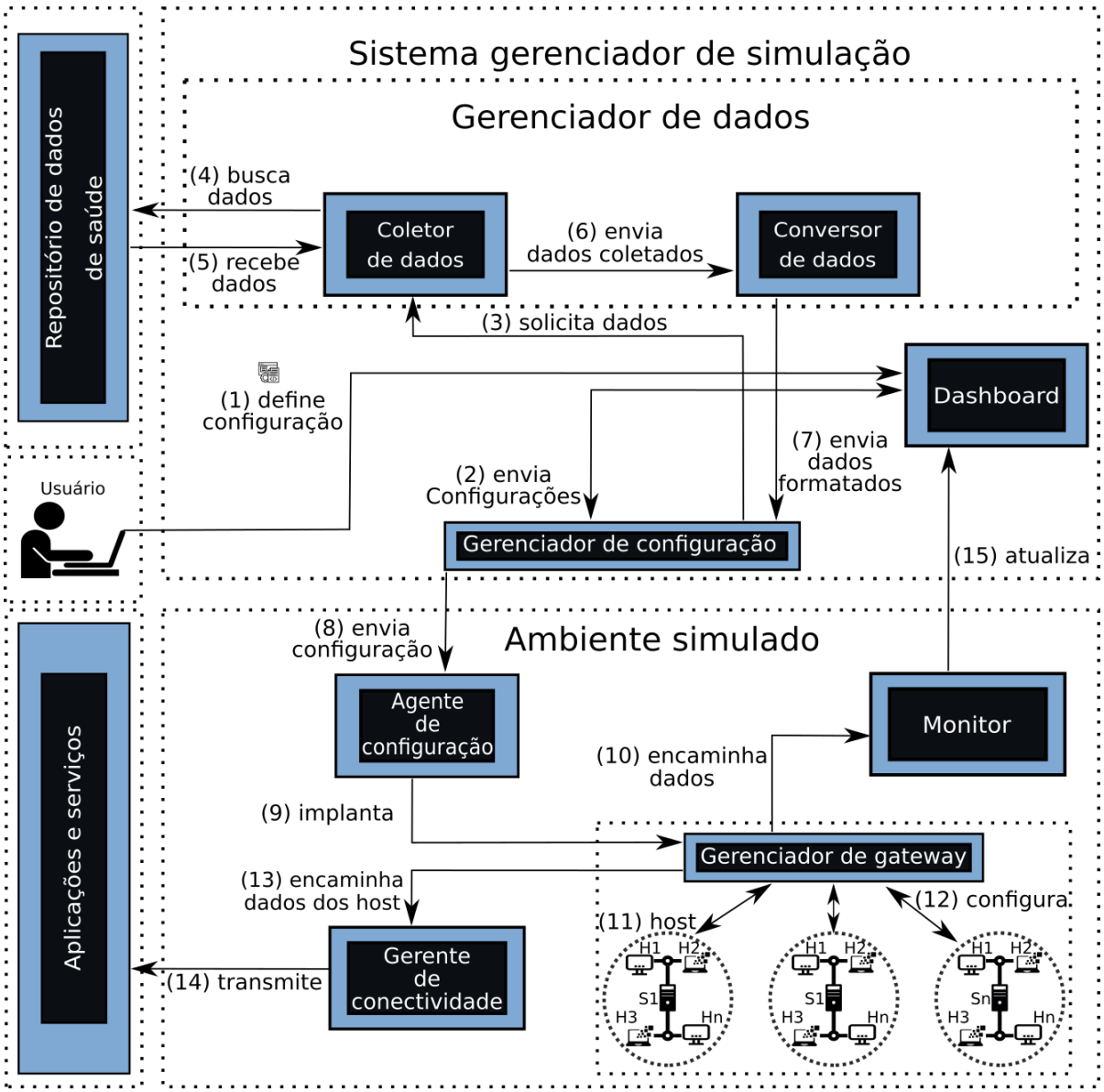
Fonte: Elaborado pelo autor

O `Gerente de conectividade` conecta o HealthSimulator com a aplicação de saúde consumidora de dados por meio uma interface de comunicação padrão. A escolha do protocolo utilizado pelo `Gerente de conectividade` dependerá inteiramente do método de comunicação suportado pela infraestrutura eHealth sob teste.

4.3 Interações na plataforma HealthSimulator

Esta seção resume a interação entre os componentes da plataforma HealthSimulator, mostrando como se dá o fluxo de dados dentro da plataforma a partir do momento em que o usuário inicia as configurações do sistema para uma simulação específica visando avaliar uma aplicação em um cenário de cuidados em domicílio. A Figura 17 ilustra a arquitetura geral do HealthSimulator destacando as interações internas entre os seus componentes.

Figura 17 – Componentes da arquitetura e suas interações



Fonte: Elaborado pelo autor

O usuário, na etapa (1), começa por definir as configurações gerais de simulação e do ambiente simulado almejado, indicando, por exemplo, o tipo de registro a ser simulado, a topologia local da rede, a quantidade de nós e os *gateways* que serão operados no módulo `Dashboard`. Após o usuário definir as configurações iniciais o `Dashboard` recebe, codifica e, na etapa (2), transmite as configurações ao componente `Gerenciador de configuração`, que é responsável por coordenar as definições do usuário, por exemplo, solicitar, na etapa (3), os registros biomédicos ao módulo `Coletor de dados`. Este, por sua vez, consulta os dados no `Repositório de dados de Saúde` na etapa (4). Em seguida, os dados localizados e recebidos na etapa (5) são encaminhados na etapa (6) para o

`Conversor de dados`, encarregado de convertê-los para um formato compreendido pela aplicação de destino. Posteriormente, na etapa (7), o módulo `Conversor de dados` envia os dados formatados para o componente `Gerenciador de configuração` que, por sua vez, recebe os registros no formato definido pelo usuário. Assim, o Gerenciador de configuração se encarrega de empacotar e armazenar os registros formatos agrupando-os ao endereço do arquivo junto as definições do usuário, por exemplo, tamanho da amostra, taxa de frequência e tempo da simulação.

Finalizadas as etapas de configuração ocorre o início da comunicação entre as camadas da plataforma, realizada entre os componentes `Gerenciador de configuração` e o `Agente de configuração`, sendo o último encarregado de receber, na etapa (8), as definições de configurações de topologia de rede do `Gerenciador de configuração` e por requisitar os registros para o início da simulação. O `Agente de configuração` também é responsável por configurar o ambiente virtual nos moldes das configurações estabelecidas pelo usuário, por meio do ajuste da quantidade de *hosts virtuais* (sensores), *switches*(*gateway*) e largura de rede. Por fim, na etapa (9), o `Agente de configuração` é incumbido pela implantação das configurações no `Gerenciador de gateway`.

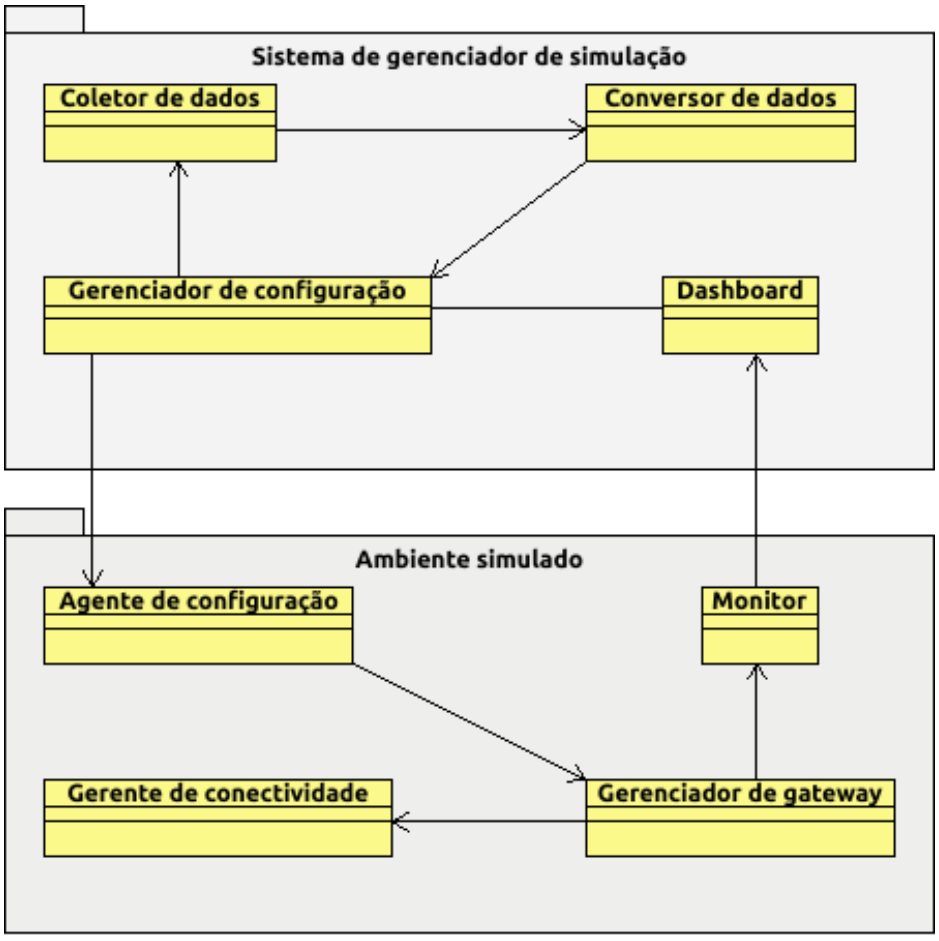
Com as configurações do ambiente virtual definidas o `Gerenciador de gateway`, na etapa (12), configura o *layout* da rede, o tráfego de informações e como os dispositivos serão conectados. Além disso, também configura os ambientes virtuais. Para isso, o gerenciador empacota uma aplicação ou o ambiente inteiro dentro de um contêiner de modo que o ambiente virtual pode se tornar portável para qualquer outro *Host* que possa oferecer informações, recursos, serviços às aplicações de Saúde na etapa (11). Assim, não haverá necessidade de ajustes individuais para o correto funcionamento da rede caso ocorra um aumento da demanda de cuidados de Saúde em virtude do aumento do número de pacientes à procura por maior acessibilidade aos cuidados de Saúde fora dos hospitais.

Após processar as simulações nos ambientes virtuais, os dados são direcionados para o `Gerenciador de gateway`, que encaminha os registros simulados do experimento para o `Gerente de conectividade` na etapa (13). Este componente recebe os dados dos ambientes simulados e os encaminha por meio do protocolo de comunicação estabelecido durante o processo de configuração feito pelo usuário. Desta forma, a `Camada de conectividade` é encarregada de transmitir os dados por meio de mensagens ou pacotes de dados do `Ambiente simulado` para as `Aplicações e serviços` de Saúde. Por fim, o componente `Aplicações e serviços` executa os serviços das aplicações de Saúde em escala gerada com os registros reais pelo simulador.

4.4 Implementação

A implementação do HealthSimulator foi dividida em 2 módulos principais: *manager*, mapeado no diagrama de classes da Figura 18 como `Sistema gerenciador de simulação`, e o *simulator*, mapeado como `Ambiente Simulado`. O objetivo é facilitar a reutilização de código do HealthSimulator em futuras atualizações. O módulo *manager* é composto pelos códigos-fontes responsáveis por gerenciar a requisição e conversão dos registros. Já o módulo *simulator*, representado na Figura 18 pelo pacote Ambiente Simulado, contém os arquivos responsáveis pelo gerenciamento e execução da simulação das aplicações eHealth analisadas.

Figura 18 – Diagrama de classe do HealthSimulator



Fonte: Elaborado pelo autor

A modularização apresentada pelo HealthSimulator permite que novos módulos possam ser agregados ao sistema conforme a necessidade do usuário. Por exemplo, o HealthSimulator pode ser adaptado para usar um novo repositório de dados de Saúde, um novo padrão de protocolo de comunicação ou mesmo um novo formato de dados de Saúde por meio da implementação das alterações correspondentes no **Coletor de dados**

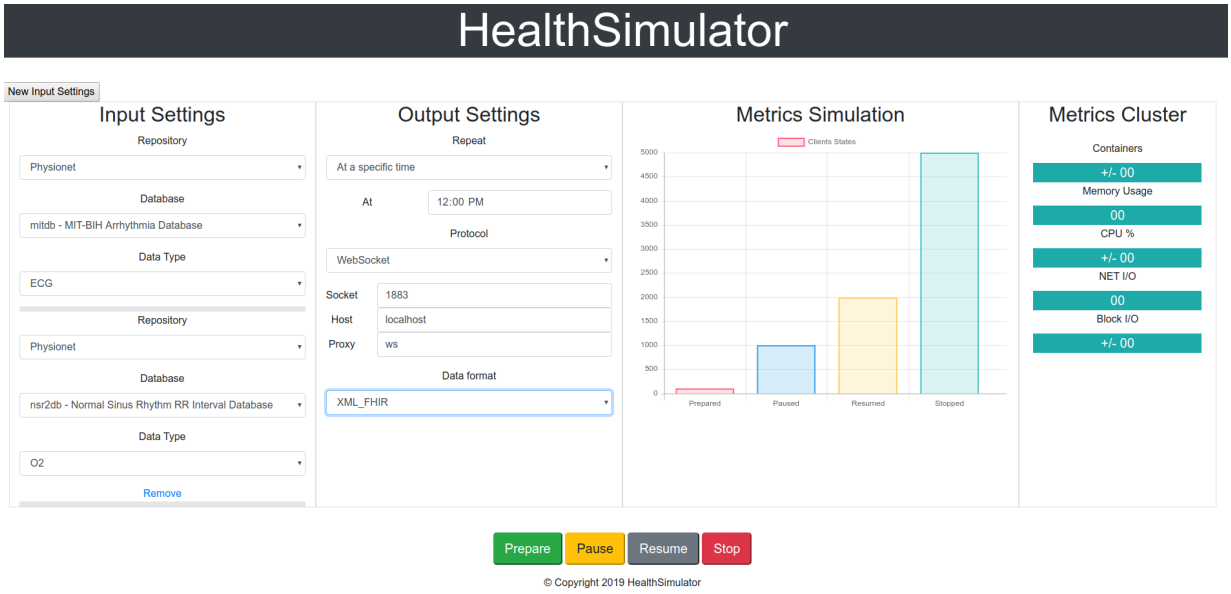
e no **Conversor de dados**. As subseções seguintes apresentam a descrição detalhadas das principais características e abstrações da implementação do HealthSimulator.

4.4.1 Interface Dashboard

A interface do HealthSimulator foi construída utilizando HTML5 e framework Flask para desenvolvimento *Web* em Python. O Flask foi escolhido devido à sua rápida inicialização, adequada a um ambiente interativo, e também por permitir criar aplicações completas e pequenas, o que é ideal para rodar em ambientes que possuam limitações de recursos. A linguagem de programação Python foi selecionada devido às características que fazem dessa tecnologia uma das mais interessantes linguagens de programação do mercado atual de desenvolvimento, sendo que algumas dessas características foram muito importantes e necessárias para o desenvolvimento deste projeto, tais como: interoperabilidade, multiplataforma, robustez, velocidade de aprendizado e simplicidade.

A interface provida pelo HealthSimulator permite a seleção e inserção do tipo de registros, descrição do repositório e banco de dados, além das configurações do formato de saída dos registros coletados dos repositórios, mediante a configuração de algumas características, tais como a frequência de envio dos registros, o tipo de protocolo de comunicação a ser estabelecido pela aplicação, o formato dos dados e, ainda, a escala.

Figura 19 – Exemplo da interface de configuração – HealthSimulator



Fonte: Elaborado pelo autor

A Figura 19 apresenta a tela inicial da interface do simulador. A configuração dos dados de entrada e também dos dados de saída é feita por meio do preenchimento dos

campos *Input Settings* e *Output Settings*, respectivamente, que representam a maneira pela qual o HealthSimulator solicita ao usuário as informações para iniciar a simulação da aplicação de Saúde sob avaliação. Vale lembrar que elementos e campos do **Dashboard** são retráteis e aparecem de acordo com a seleção do usuário. Assim, para executar uma simulação, basta o usuário preencher e selecionar todos os campos e clicar no botão *Prepare* para que as configurações por ele definidas sejam enviadas diretamente para o `Gerenciador de configurações`, que é materializado por meio da instanciação da classe ManagerConfig(data_config) passando um arquivo JSON com todas as configurações. O acionamento do botão *Stop* interrompe todos os processos do simulador, a qualquer momento.

Toda a implementação do **Dashboard** se encontra na pasta *server*. Este diretório é responsável por concentrar todos arquivos do **Dashboard**, tais como *api*, arquivos estáticos da página, *template*, além de componentes como urls e views. O HealthSimulator permite a construção de novos módulos, portanto, a pasta *api* visa armazenar todas as versões da *interface* do simulador. Logo, sempre que houver uma atualização ou a criação de um novo componente para o **Dashboard** será versionado e armazenado neste diretório, possibilitando assim a utilização de múltiplas versões deste componente caso seja necessário o uso de versões diferentes para uma determinada aplicação de saúde.

Buscou-se, durante implementação do HealthSimulator, tornar o simulador um ambiente inteiramente funcional via *Web*, a fim de permitir que o usuário realize testes práticos em sua aplicação de Saúde em qualquer lugar e tempo, utilizando dados reais e já validados por outros pesquisadores. Tal iniciativa viabiliza a economia de tempo gasto com a preparação do ambiente de codificação necessário à resolução das atividades propostas e torna o ambiente mais próximo do real.

4.4.2 Gerenciador de configuração

O **Gerenciador de configuração** simplifica a interação entre o *manager*, responsável pelo gerenciamento da simulação, e o *simulator*, o ambiente simulado. Isso é feito mediante coordenação dos elementos situados no módulo *manager*, o Conversor de dados e o Coletor de dados.

O **Gerenciador de configuração** foi implementado pelo script **manager()**, alocado no diretório *base*. O script de implementação do **manager()** é formado pela classe **ManagerConfig()**, mostrada na Figura 20. Esta classe recebe como parâmetro de entrada um arquivo em JSON proveniente do **Dashboard** contendo todas as configurações selecionadas e marcadas pelo usuário antes da execução do simulador pelo acionamento do botão *Prepare*. O parâmetro data_config recebido pela classe **ManagerConfig()** contém tanto informações de entrada quanto de saída. Como já mencionado, a comunicação com o ambiente simulado pode ocorrer por meio de quatro estados diferentes: *prepare*, *pause*,

resume e *stop*,representados pelos respectivos ícones na interface de configuração.

Figura 20 – Implementação do Gerente de configuração

```
class ManagerConfig():

    def __init__(self, data_config):
        self.data_config = data_config
        settings.REPOSITORY = self.data_config["config_manager"]["repository"]

    def simulation_state(self):…

    def prepare_simulation(self):…

    def pause_simulation(self):…

    def resume_simulation(self):…

    def stop_simulation(self):…
```

Fonte: Elaborado pelo autor

Uma tarefa deste módulo é permitir que as configurações do usuário sejam repassadas para o módulo *simulator*, que representa o ambiente simulado. Para esta tarefa optou-se pelo envio das configurações do simulador por meio de mensagens MQTT. Para tal, os dois módulos assinam o tópico /CONFIG no endereço padrão e na porta 1883. Dessa forma, o componente **Gerenciador de configuração** estabelece comunicação com o componente `Agent` do módulo *simulator*. Num primeiro momento, os dados das configurações e os registros dos repositórios eram encaminhados todos via MQTT; no entanto, notou-se que para determinada quantidade de registros os pacotes eram perdidos pelo gerenciador do MQTT PAHO. Deste modo, optou-se por manter o envio das configurações por esse método e enviar os registros por meio de uma requisição a um servidor HTTP feita pelo **Agente de configuração** no módulo *simulator*. O **Gerenciador de configuração** recebe as informações e as executa.

Após a execução das etapas de coleta e conversão, os registros são encaminhados para o componente de configuração, que os empacota e os nomeia com um número UUID único. Por sua vez, o **Gerenciador de configuração** empacota os dados de configurações de saída dos registros em um arquivo JSON contendo os dados de configuração selecionados pelo usuário e, ainda, o endereço UUID onde estão os registros convertidos e compactados para serem simulados no módulo *simulator*.

Uma importante característica do **Gerenciador de configuração** é a sua flexibilidade para manipular e gerenciar as configurações e os diversos tipos de interações entre as aplicações e o simulador. Em última instância, é o **Gerenciador de configuração** quem realiza as atividades pertinentes à disponibilização e controle da infraestrutura e do ambiente de configuração que dará suporte às simulações.

4.4.3 Obtenção e conversão dos dados

O Gerenciador de dados disponibiliza uma API que representa as interfaces fornecidas e requeridas para utilização dos repositórios de dados. Esta API permite ao usuário o acesso e o envio dos registros biomédicos dos pacientes para validar a sua aplicação.

Os registros biomédicos para a realização da simulação podem ser obtidos pelo HealthSimulator por meio de *Web Services* disponibilizados pelos repositórios de dados ou desenvolvida pelo usuário. A API funciona como uma ponte entre o repositório e o simulador, sendo implementada pelo componente **Coletor de dados**.

O componente **Coletor de dados** é instanciado para cada repositório por meio da classe **RepositoryDataLoader()**, que é composta pelos seguintes métodos: **list_databases()**, **check_database()**, **download()** e **get_records()**. Esses são os métodos básicos para o funcionamento do simulador, os quais devem ser implementados de acordo com a inclusão dos novos repositórios de dados.

A Figura 21 ilustra os métodos obrigatórios do **Coletor de dados** do HealthSimulator. O método **list_databases()** é responsável por listar todos os bancos de dados disponíveis no repositório escolhido pelo usuário; o método **check_database()** é encarregado de verificar se o banco de dados solicitado pertence ao repositório determinado pelo usuário e o método **download()** acessa e buscar os dados requisitados no repositório especificado pelo usuário, e retorna os registros para o simulador.

Figura 21 – Implementação do coletor de registros – HealthSimulator

```
class RepositoryDataLoader():

    def __init__(self):
        pass

    def list_databases(self):
        raise NotImplementedError()

    def check_database(self, name):
        raise NotImplementedError()

    def download(self, name):
        raise NotImplementedError()

    def get_records(self, name):
        raise NotImplementedError()
```

Fonte: Elaborado pelo autor

O diretório *repository* é constituído pelas implementações das classes abstratas utilizadas *loader* e *converter*, respectivamente **Coletor de dados** e **Conversor de dados**.

Para cada repositório de dados implementado pelo HealthSimulator será criado um novo diretório com o nome do repositório desejado, contendo os arquivos de implementação daquele repositório. Deste modo, para que ocorra a comunicação e conversão apropriadas com o simulador, novos repositórios devem obrigatoriamente implementar as classes abstratas descritas anteriormente.

A Figura 22 ilustra a implementação da classe **PhysionetDataLoader()**, referente ao repositório de dados PhysioNet, implementado neste trabalho. A classe **PhysionetDataLoader()** implementa e herda automaticamente do **RepositoryDataLoader()** todos os atributos, propriedades e métodos da classe pai. Além disso, outros métodos utilizados pelo PhysioNet também são implementados nesta mesma classe. Vale salientar que comunicação com o repositório PhysioNet foi feito por meio da utilização da biblioteca multiplataforma WFDB em Python de código aberto. Em síntese, o PhysioNet foi escolhido como primeiro repositório a ser implementado pelo HealthSimulator devido a sua grande documentação e facilidade de implementação.

Figura 22 – Implementação do coletor de registros do repositório PhysioNet

```
class PhysionetDataLoader(RepositoryDataLoader):

    def __init__(self):⋯

    def _load_databases(self):⋯

    def list_databases(self):⋯

    def check_database(self, nameDB):⋯

    def download(self, nameDB):⋯

    def get_records(self, nameDB):⋯

    def verify_records_exist(self, database):⋯

    def download_record(self, database, sampFrom=None, sampTo=None):⋯
```

Fonte: Elaborado pelo autor

Após capturar os registros de saúde disponibilizados pelo repositório, o componente **Conversor de dados** entra em ação para extrair os registros e os metadados do banco selecionado e adequá-los para o formato compatível e aceito pela aplicação de destino, tendo em vista que os dados retornados do repositório podem estar em diferentes formatos. A implementação do **Conversor de dados** padroniza, portanto, a manipulação dos registros pelo simulador. Este componente foi implementado pela classe **DataConverter()**, como apresentado na Figura 23. O **DataConverter()** é um componente abstrato. Este script é composto pelo método **convert_record()** e pelo **convert()**, um método obrigatório e

que deve ser implementado caso ocorra a adição de um novo repositório e/ou formato de dados. Este tipo de implementação foi feita levando em conta a demanda da área de Saúde, em que os padrões de dados são importantes e estão em constante mudança. Portanto, por ser uma classe abstrata, pode ser implementada por padrões de dados de Saúde existentes ou por novos padrões, por exemplo, o formato HL7 FHIR.

Figura 23 – Implementação do conversor de registros – HealthSimulator

```
class DataConverter():

    def convert(self, records):
        for record in records:
            self.convert_record(record)

    def convert_record(self, record):
        raise NotImplementedError()
```

Fonte: Elaborado pelo autor

A Figura 24 apresenta a implementação do componente **Conversor de dados** no padrão de dados FHIR. Este padrão é implementado pelo simulador e pode ser disponibilizado em diferentes formatos, por exemplo, JSON, XML e YAML. Considerando a complexidade do ambiente de Saúde, o uso de diferentes formatos de dados pode ser útil. Todos protocolos utilizados por aplicações IoT possuem algum tipo de vantagem e desvantagem em relação ao seu uso, seja ela semântica, estática e de implementação. Portanto, sob o aspecto de uma estrutura lógica e dependendo da complexidade de um determinado cenário, pode ser que a estrutura de dados a ser manipulada seja mais simples de trabalhar em um determinado formato. Por exemplo, a manipulação de arquivos com vários campos é mais simples em JSON do que em XML pois JSON se baseia na estrutura chave-valor em vez da estrutura de árvore utilizado pelo XML. O método **convert()** implementado pela classe **FhirConverter(DataConverter)** do HealthSimulator transforma um registro por vez em um formato interpretável pela aplicação do usuário, ao passo que o método **convert_record()** é capaz de receber dados em diferentes formatos e retornar uma lista de registros padronizados no formato determinado pelo usuário de uma determinada base de dados. Resumidamente, os métodos da classe conversor recebem como parâmetro os registros a serem transformados e um arquivo contendo as configurações de conversão dos dados e, em seguida, torna o registro acessível para os outros módulos do simulador e também pela aplicação a ser testada.

A classe **PhysionetFhirConverter(FhirConverter)** implementa a classe abstrata **FhirConverter(DataConverter)** que, por sua vez, implementa **DataConverter()** e os seus métodos. Ou seja, DataConverter é uma superclasse abstrata a qual pode ser utilizada para implementar outros formatos de dados além do escolhido por este

Figura 24 – Implementação do conversor de registros para o formato FHIR

```python
from converter.base import DataConverter


class FhirConverter(DataConverter):

    def convert(self, records, manager):
        list_records = []
        for record in records:
            if(self.convert_record(record, manager) is not None):
                list_records.append(self.convert_record(record, manager))

        print("Data converted to FHIR!")
        return list_records

    def convert_record(self, record, manager):
        raise NotImplementedError()
```

Fonte: Elaborado pelo autor

trabalho, o HL7 FHIR. Esta classe implementa o método herdado **convert_record()** e, além disso, possui diferentes métodos que gerenciam a padronização dos dados de entrada no formato especificado.

Figura 25 – Implementação do conversor de registros do PhysioNet para o formato FHIR

```python
class PhysionetFhirConverter(FhirConverter):
    id_record = None
    signals = None
    fields = None
    manager = None

    def read_dat_file(self, nameDB):...
    # retorna a parte inicial do dict fhir

    def fhir_template_base(self):...

    def fhir_template_component(self, list_signal, indice):...

    def partition_data_time(self, indice):...

    def create_fhir_file(self):...

    def convert_record(self, record_db, manager):...
```

Fonte: Elaborado pelo autor

As configurações do Gerenciador de dados estão armazenadas na pasta *repository* do lado do gerenciador denominado na implementação de *manager*. Diante da possibilidade

de expansão de novos repositórios, protocolos e padrões de dados, optou-se pela separação dos componentes de coletor e conversor por repositório de dados. Assim sendo, a adição de novos componentes é simplificada por meio da implementação de métodos especificados por uma classe abstrata já definida. A principal motivação por trás desta escolha é encontrar soluções para facilitar a integração de aplicações de monitoramento de Saúde e, suportar a reutilização dos componentes desenvolvidos em diferentes aplicações.

4.4.4 Agent

O componente **Agente de configuração**, responsável por gerenciar as configurações de saída do simulador no módulo *simulator*, é implementado pela classe **AgentConfig()**, apresentada na Figura 26. O script é formado pelo método **start_mqtt_agent()**, que recebe as configurações, **process_message()**, que gerencia as requisições e também controla o método **simulation_state()**, responsável por coordenar os quatro estados do simulador descritos na Subseção 4.4.2.

Figura 26 – Implementação do Agente de configuração – HealthSimulator

```
class AgentConfig():
    state = None
    config = None
    data_list = None
    uuid = None
    client = None
    loop = None

    def __init__(self, loop):...

    async def misc_loop(self):...

    async def start_mqtt_agent(self):...

    async def process_message(self, msg):...

    async def simulation_state(self):...

    async def prepare_simulation(self):...

    def pause_simulation(self):...

    def resume_simulation(self):...

    def stop_simulation(self):...

    async def run(self):...
```

Fonte: Elaborado pelo autor

Mediante o recebimento da mensagem com o arquivo de configuração enviada via MQTT para a classe *AgentConfig()*, o método **process_message()** começa a executar o processo de requisição dos dados, passando a ser responsável pelas atividades descritas pelo **Agente de configuração**.

Inicialmente, as configurações são recebidas pelo **start_mqtt_agent()** e descompactada para um arquivo `data_config` no formato JSON, ilustrado na Figura 27. Em seguida, o **process_message()** é invocado para realizar a requisição do arquivo nomeado com o número UUID e armazenado no *manager* com os registros biomédicos. Vale ressaltar que a requisição dos registros convertidos e armazenados compactados é feita via servidor *WebSocket* pelo **Agente de configuração**. Para concluir o processamento, o método **simulation_state()** é disparado após todo o procedimento de requisição dos dados para determinar o estado atual do simulador e determinar a ocorrência do próximo evento.

Figura 27 – Arquivo de configuração – HealthSimulator

```
{
    "state": 1,
    "config_manager": {
        "repository": "Physionet",
        "database": "mitdb",
        "dataType": "ECG"
    },
    "config_simulator": {
        "connectNumber": "1",
        "format": "json",
        "protocol": {
            "protocol": "mqtt",
            "host": "localhost",
            "topic": "/DATA2",
            "port": "1883"
        },
        "repeat": "* * * * * * * 1 * *"
    }
}
```

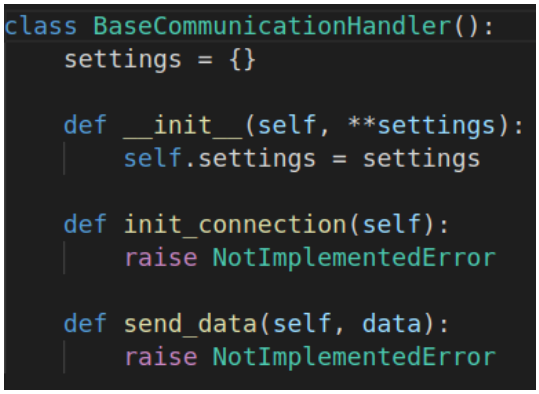
Fonte: Elaborado pelo autor

Também faz parte do componente **Agente de configuração** a biblioteca *scheduler* do Python. Esta biblioteca provê a capacidade de agendamento de execução dos estados do simulador. Compete ao *scheduler* a responsabilidade de disparar e coordenar a execução dos outros componentes a fim de garantir a existência e a consistência da simulação. É neste componente que ocorre a iniciação do processo simulação por meio do agendamento e controle de execução de cada módulo do HealthSimulator.

4.4.5 Conectividade

O componente **Gerente de conectividade** descrito na Subseção 4.2.6 é implementado pela classe **BaseCommunicationHandler()** do pacote *comm*, representado na Figura 28. Este componente é encarregado de transferir os registros para a aplicação de monitoramento simulando, assim, o envio de dados reais. A forma de envio se assemelha aos métodos utilizados no ambiente real de monitoramento, ou seja, o envio periódico de registros. Nesta comunicação, os protocolos comumente utilizados na literatura para monitoramento de pacientes são empregados pelo HealthSimulator.

Figura 28 – Implementação da classe abstrata de comunicação – HealthSimulator



Fonte: Elaborado pelo autor

A versão atual do HealthSimulator é capaz de transferir registros coletados do repositório de Saúde mediante a utilização dos seguintes protocolos de comunicação: MQTT, CoAP e *WebSocket*, os dois primeiros protocolos típicos de ambientes de IoT, composto de dispositivos de recursos computacionais limitados. Além disso, devido à natureza do ambiente de Saúde pode ocorrer a necessidade de adição novos protocolos de comunicação. Dessa forma, basta implementar os métodos obrigatórios da classe abstrata **BaseCommunicationHandler()** de acordo com os requisitos e características do protocolo desejado.

O componente **Gerente de conectividade** fornece, portanto, um ambiente de comunicação uniforme e baseado em padrões para a simulação das aplicações de Saúde e simplifica o processo de implementação de novos protocolos de comunicação. Por exemplo, a classe **CommunicationHandler(BaseCommunicationHandler)** mostrada na Figura 29 apresenta a implementação concreta da classe abstrata **BaseCommunicationHandler()** pelo protocolo MQTT. Deste modo, todos os métodos necessários para comunicação com aplicação de monitoramento são herdados e implementados pelo protocolo de interesse.

Figura 29 – Implementação da classe abstrata de comunicação para o protocolo MQTT – HealthSimulator

```
class CommunicationHandler(BaseCommunicationHandler):
    client = None
    url = None

    async def init_connection(self, adress):
        self.client = mqtt.Client()
        self.url = self.settings.get(
            "url", urlparse(adress))
        self.client.connect(self.url.hostname or "localhost",
                            self.url.port or 1883)

    async def send_data(self, data):
        print("sending data")
        topic = self.url.path
        self.client.publish(topic, str(data))
```

Fonte: Elaborado pelo autor

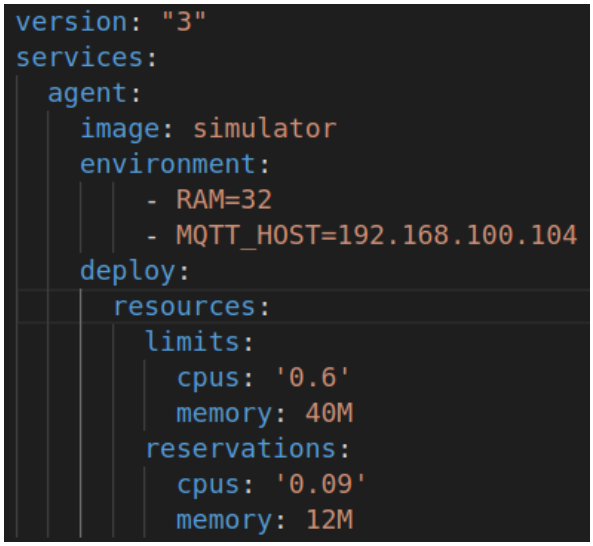
4.4.6 Escalabilidade

Como visto, o **Gerente de gateway** é o componente da arquitetura responsável por tratar questões de escalabilidade. O HealthSimulator aborda esta questão mediante a utilização da ferramenta Docker na implementação do **Gerente de gateway**. O uso de contêineres Docker fornece uma maneira rápida e fácil de criar e implantar novos aplicativos no ambiente de teste e possibilita o acréscimo no número de nós simulando dispositivos enviando dados para aplicação de monitoramento de Saúde sendo testada. Com o uso dessa ferramenta busca-se amenizar o problema de escalabilidade, fornecendo meios para dimensionar partes de um aplicativo que causam contenção e gargalos, já que o Docker possibilita alterar as limitações de recursos em tempo de execução para um contêiner, como o tempo da CPU e memória disponível.

O sistema do Docker permite implementar e executar imagens de contêineres em máquinas distribuídas. Este esquema possibilita ao HealthSimulator executar aplicações conteinerizadas em escala, já que da mesma forma que é executado um contêiner único, o ambiente Docker pode iniciar um processo replicado e distribuído. Durante a prototipagem do simulador e para a realização dos testes de integração foi criada uma imagem para ser utilizada em um contêiner Docker (Figura 30), simulando um agente que recebe os registros de saúde e os encaminha para aplicação de monitoramento.

A funcionalidade de escalabilidade oferecida pelo HealthSimulator para avaliar o uso da aplicação sobre demanda é provido por meio do conjunto de ferramentas e serviços do Docker, sendo assim, utilizando uma imagem do simulador o usuário pode

Figura 30 – Exemplo de configuração do Docker-Compose – HealthSimulator



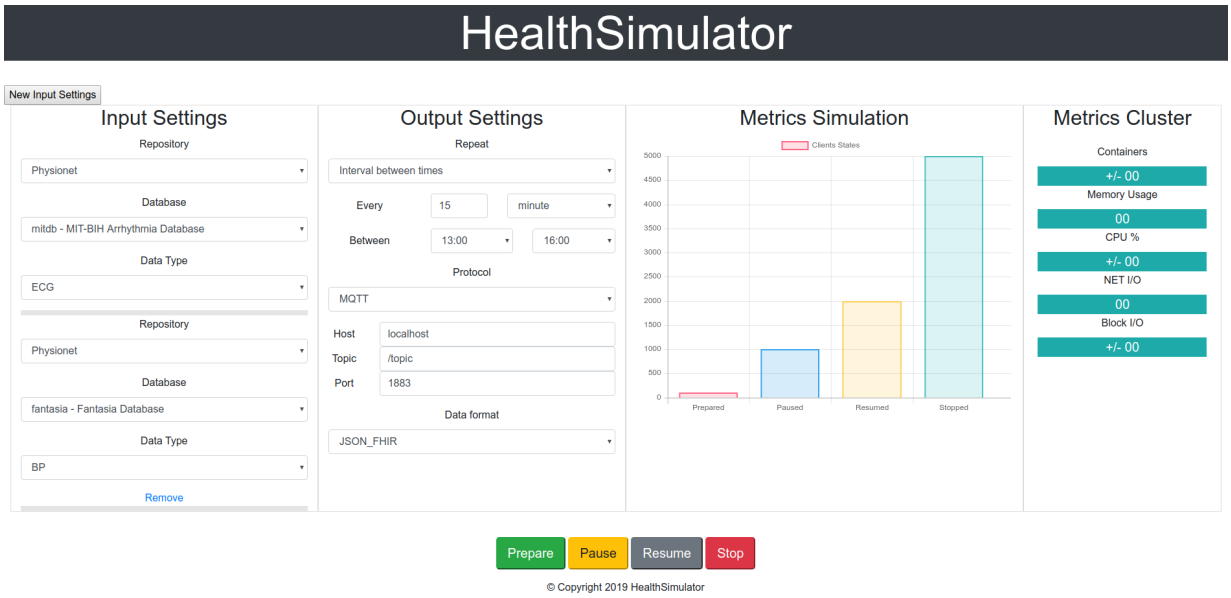
Fonte: Elaborado pelo autor

implantar em sua própria infraestrutura e aprimorar com seu conjunto e modelo de dados personalizado. Com base na quantidade de carga e recursos necessários, o usuário pode escalar e avaliar sua aplicação em desenvolvimento ou durante a implantação usando as ferramentas fornecidas pela tecnologia gerenciador de contêineres Docker. Por exemplo, se o serviço do HealthSimulator demandar uma carga maior de envio de dados para atender a solicitações de serviços, ou seja, a aplicação eHealth, o usuário poderá aumentar o número de instâncias da API com um único comando (por exemplo, docker run -it 'health-simulator' -e ENV_NUM_PROCS=10). Assim, distribuindo o envio e a demanda dos dados em dez novas instâncias do Docker (*hosts* de Docker) sem nenhum esforço adicional do usuário para orquestrar e configurar esses serviços.

4.4.7 Configurando o HealthSimulator

As configurações do simulador HealthSimulator são inseridas pelo usuário por meio da interface *Web*, dividida em quatro colunas principais. As duas primeiras são responsáveis pela configuração e as duas últimas pela visualização das métricas de simulação. Sendo assim, a primeira coluna, *Input Settings*, contém as configurações de entrada do simulador; a segunda coluna, intitulada de *Output Settings*, possui as configurações de saída. Além disso, a interface gráfica apresenta as informações da simulação por meio das métricas de simulação (*Metrics Simulation*) e por meio das métricas do gerenciador de contêineres (*Metrics Cluster*), na terceira e quarta colunas da interface, respectivamente. Em seguida, abaixo da área de configuração e de acompanhamento, seguem os quatro botões de estados do simulador: *Prepare*, *Pause*, *Resume* e o *Stop*.

Figura 31 – Exemplo de uso da interface de configuração HealthSimulator



Fonte: Elaborado pelo autor

A Figura 31 ilustra um exemplo de configuração dos parâmetros de entrada e saída na interface do HealthSimulator, bem como o resultado (*status*) de uma simulação. O *Input Settings*, na Coluna 1 requer que três parâmetros de entrada sejam inseridos pelo usuário durante a configuração, entre eles o repositório de dados de Saúde disponíveis, o banco de dados daquele repositório e ainda o tipo de dado biomédico disponibilizado pela base de dados. Já a segunda coluna, *Output Settings*, permite configurar como o simulador deve se comunicar com a aplicação a ser avaliada, por meio da inserção do tipo de repetição do envio dos dados para a aplicação, o protocolo de comunicação utilizado pela aplicação a ser avaliada (MQTT, CoAP ou *WebSocket*) e, ainda, o formato de saída dos dados suportados pela aplicação eHealth, ou seja, a padronização. Vale ressaltar que o tipo de repetição e os protocolos comunicação, traz como configuração padrão do simulador três tipos de configurações comumente utilizadas no cenário de monitoramento remoto cada um, cobrindo assim um maior número de possíveis cenários de configuração. Os parâmetros das duas primeiras colunas são recebidos pelo `Gerenciador de Configuração` e manipulados de acordo com as configurações definidas no *Input* e *Output Settings*, em seguida são enviados para o `Agent` encarregado de configurar no simulador os dados de saída definidos.

As métricas apresentadas pela Coluna 3 e 4 da interface do simulador, refere-se às métricas de clientes (nós) conectados e de uso de recursos para simulação na devida ordem. *Metrics Simulation* funciona de forma interativa, apresentando o número de clientes conectados, pausados, retomados e os nós parados durante a simulação. Na Coluna 4, *Metrics Cluster* traz algumas informações importantes do gerenciador de contêineres, como

o número de contêineres sendo executados no momento, o uso de memória e CPU, além de informações de entrada/saída de rede disponibilizada pelo contêiner correspondente e informações do total de *bytes* gravados e lidos no contêiner. O componente `Monitor` descrito na arquitetura é responsável por atualizar e sincronizar as mudanças relativas às métricas observadas no `Dashboard`.

4.5 Considerações finais do capítulo

Este capítulo apresentou o projeto da arquitetura do HealthSimulator, um ambiente projetado para avaliar infraestruturas de eHealth em ambiente AAL. O capítulo descreve ainda a implementação de um primeiro protótipo desta arquitetura, usando como base tecnologias padronizadas de comunicação e estruturação de dados. A validação inicial do ambiente de simulação desenvolvido é feita por meio de uma prova de conceito, introduzida no Capítulo 5 a seguir.

5 Prova de conceito

Para demonstrar a operação da plataforma e validar o funcionamento dos componentes da arquitetura foi desenvolvida uma prova de conceito, que consiste basicamente na realização de dois testes: (i) teste de acesso a um repositório de dados biomédicos online e envio destes dados para uma aplicação simples, que foi desenvolvida especificamente para consumir estes dados; e (ii) testes de escalabilidade da plataforma, cujo objetivo é verificar o comportamento do simulador quando número de pacientes aumenta muito em escala, isto é, se ele simula adequadamente o aumento do volume de dados a ser encaminhado para a aplicação.

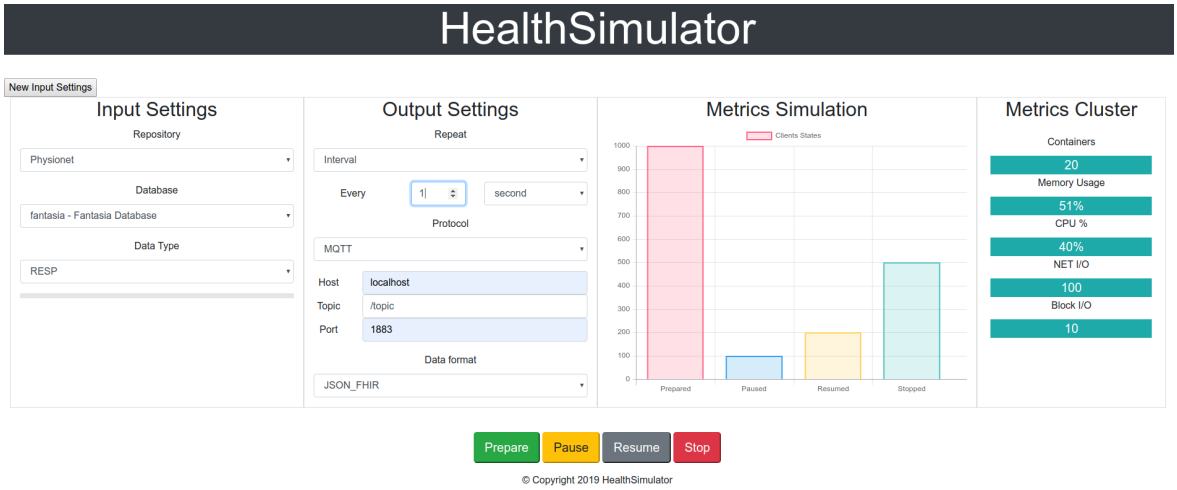
A máquina usada para a realização da prova de conceito foi equipada com um processador Intel Core i7, modelo 8550U 8ª Geração, de 1.8GHz (até 4.0GHz Max Turbo com 8 MB cache) e uma memória RAM de 8Gb. O sistema operacional utilizado foi o Ubuntu, versão 19.04 de 64 bits.

5.1 Teste de acesso a um repositório com visualização dos dados gerados

Neste primeiro teste, a plataforma simula a coleta, a conversão e o envio dos dados via protocolo MQTT para uma aplicação concebida com a ferramenta de programação Node-Red, que recebe e apresenta os registros biomédicos dos pacientes por meio de gráficos para análise do profissional de Saúde. Para a composição deste cenário de teste foram utilizados sinais de ECG e pressão arterial obtidos a partir da base Fantasia do PhysioNet (MCCONALOGUE; DAVIS; CONNOLLY, 2018). Esta base é composta por registros de 40 indivíduos, 20 jovens (21-34 anos) e 20 idosos (68-85 anos), entre eles um número igual de sujeitos masculinos e femininos. Este registros são organizados em ordem sequencial com relação ao tempo de coleta e há apenas um registro de duas horas para cada sujeito: os sinais contínuos de ECG, respiração e pressão arterial digitalizados a 250 Hz. Cada batimento cardíaco foi anotado utilizando um algoritmo automatizado de detecção de arritmia e cada anotação de batimento foi verificada por inspeção visual. Além disso, foram utilizados também dados de temperatura da base **sleep-edf** (Sleep-EDF *Database* v1.0.0), que contém 197 de sinais durante o sono. Os registros estão distribuídos entre EEG, EOG, EMG, respiração e temperatura corporal, sendo este último tipo de sinal utilizado na prova de conceito.

O teste foi conduzido em três etapas. As duas primeiras referem-se à configuração do simulador e a terceira descreve o processo de simulação em si. Estas etapas são comentadas

Figura 32 – Configuração do HealthSimulator para o Teste 1



Fonte: Elaborado pelo autor

a seguir.

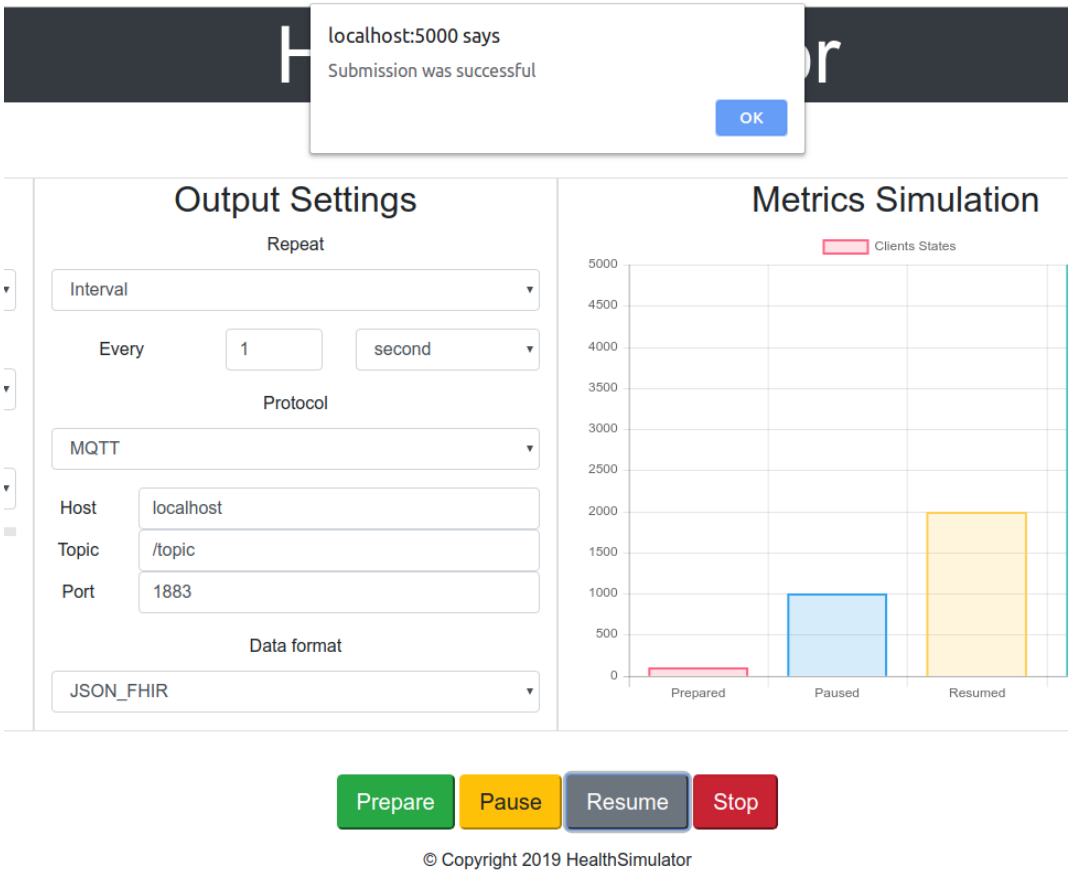
Etapa1. A *interface* principal do simulador, representada na Figura 32, foi utilizada para inserir as configurações de entrada e saída desejadas na sessão de simulação. As configurações de saída utilizadas durante a avaliação foram as mesmas para os três tipos de sinais observados. Deste modo, foram configurados os seguintes parâmetros no simulador: intervalo de um segundo para o envio dos registros; protocolo de comunicação MQTT utilizando o endereço *localhost* e porta 1883; e formato FHIR em JSON para padronizar os dados.

Vale notar que todas as informações dos dois formulários apresentados na 32 são obrigatórias, tendo em vista que a disposição dos campos pode ser alterada de acordo com a configuração anterior. O usuário dá então início à sessão de simulação que, ao seu final, trará como resultado os registro de ECG, temperatura corporal e pressão arterial do paciente, apresentados de forma gráfica pela aplicação Node-Red que simula, por sua vez, o monitoramento remoto dos sinais vitais dos pacientes.

Terminada a entrada de dados o simulador verifica se todos os campos obrigatórios estão devidamente selecionados e preenchidos. Caso todos eles estejam corretos é lançada uma mensagem de sucesso, sinalizando ao usuário o início da simulação, como mostra a Figura 33.

As configurações preenchidas pelo usuário por meio do formulário *Web* são convertidas e armazenadas no arquivo de configuração "data_config.json". Este arquivo é utilizado para instanciar a classe **ManagerConfig()**, que corresponde ao elemento `Gerenciador de Configuração` descrito na arquitetura. Por sua vez, a classe **ManagerConfig()** inicia

Figura 33 – Mensagem de sucesso ao iniciar uma simulação do HealthSimulator



Fonte: Elaborado pelo autor

a requisição dos dados à classe **PhysionetDataLoader()** do `Gerenciador de Dados` que, logo em seguida, válida a requisição.

Internamente, a validação das configurações ocorre da seguinte maneira: a classe **PhysionetDataLoader()** utiliza o método **check_database()** para verificar se o registros estão disponíveis no repositório selecionado e, caso existam e os arquivos ainda não estejam locais no diretório de repositórios do HealthSimulator, é efetuado o *download* direto da base de dados utilizando a API do repositório selecionado para o diretório **data_source**, que se encontra na pasta de dados do simulador.

Etapa2. Após a etapa de requisição dos dados do repositório PhysioNet, os mesmos são convertidos para o padrão de dados de Saúde selecionado, no caso o "FHIR" e, em seguida enviados para o módulo **simulator**. Durante os testes, devido ao tamanho dos pacotes dos registros coletados e as limitações existentes nos protocolos IoT, optou-se pelo envio em separado dos registros coletados das configurações de saída, ou seja, os registros foram empacotados e receberam um identificador UUID para serem armazenados localmente no lado gerenciador do HealthSimulator para, posteriormente, serem baixados pelo módulo

simulator via requisição ao *WebSocket* disponibilizado para esse fim. Vale notar que o processo de requisição dos dados somente ocorre após o identificador UUID ser concatenado e encaminhado para o módulo **simulator** junto aos metadados de configurações via MQTT.

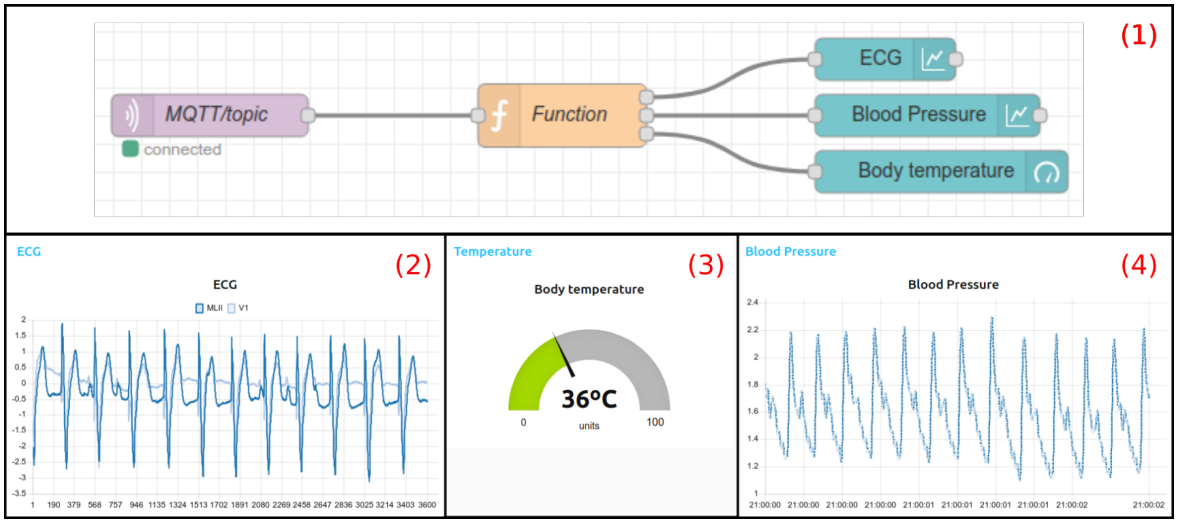
O módulo **simulator**, que fica aguardando os dados de configuração, ao recebê-los verifica o status de configuração atual selecionado. Caso seja o estado *prepare* o `Agente de Configuração` faz a requisição dos dados por meio do acesso à URL com o UUID específico. Logo após a requisição, os registros são armazenados localmente no diretório `data`, situado no módulo **simulator**. O `Agente de Configuração` inicia então a classe **SimulatorManager()**, passando os dados de configuração e os registros.

Etapa3. Nesta etapa, o HealthSimulator envia os registros e os dados de configuração da simulação para o ambiente de simulação. Para tal, a classe **SimulatorManager()** inicia a comunicação via MQTT com a aplicação Node-Red por meio dos dados de configuração de saída pré estabelecidos. Tanto o banco de dados `Fantasia` quanto o **sleep-edf** contém vários registros identificados por um ID único, sendo assim, para cada interação com a aplicação Node-Red é encaminhado um registro contendo milhares de dados biomédicos com o tipo de sinal selecionado e as observações daquele registro contidas em um arquivo JSON no padrão FHIR do tipo *Observation*. Vale destacar que a base Fantasia e a **sleep-edf** foram escolhidas por disponibilizar diferentes tipos de sinais coletados de pacientes com idades e gêneros distintos, o que amplia as possibilidades de análise por parte do profissional de Saúde.

Para controlar o fluxo de dados no Node-Red foram configurados três nós, mostrados na Figura 34(1). O primeiro nó, **MQTT/topic**, é responsável por receber os registros do HealthSimulator via MQTT através da assinatura do tópico **/topic**; o segundo nó, denominado de **Function**, é responsável por estruturar e processar os dados do registro para o formato de entrada de cada um dos três nós de `Dashboard` adjacentes, ou seja, por direcionar e formatar o fluxo de dados para cada nó. Para apresentar os dados da base selecionada, foram utilizados três nós de `Dashboard` para apresentar os sinais disponíveis nessa base de dados, identificados por `ECG`, `Blood Pressure` e `Body temperature`, respectivamente. Os três nós são encarregados por apresentar os sinais dos registros em um gráfico de linha demonstrando os sinais de ECG (2) e Blood Pressure (4), além do gráfico radial dos níveis de temperatura (3), conforme mostrado na Figura 34. Desta forma, os três tipos de nós compõem o fluxo de dados do Node-Red utilizado para simular uma aplicação de Saúde recebendo os registros biomédicos proveniente do HealthSimulator.

O processo de configuração do HealthSimulator aqui descrito para simular a geração de dados para uma aplicação Node-Red simples, como a apresentada, pode ser replicado para avaliar uma aplicação real de monitoramento de paciente, mais complexa, que manipula uma variedade mais diversa de dados. A aplicação desenvolvida mostra que, do ponto de vista funcional, a arquitetura proposta fornece o suporte necessário de geração

Figura 34 – Fluxo de dados do Node-Red



Fonte: Elaborado pelo autor

de dados de pacientes, permitindo ao desenvolvedor ou ao profissional de Saúde verificar o funcionamento da sua aplicação eHealth em um estágio inicial de desenvolvimento, ou mesmo avaliar a reação da aplicação a algum evento específico, tendo em vista que os dados utilizados possuem anotações de possíveis anomalias.

5.2 Teste de escalabilidade do simulador

O objetivo deste teste é demonstrar a elasticidade do HealthSimulator, através da simulação de vários *gateways* que enviam os dados para um servidor MQTT em intervalos preestabelecidos de tempo. Para a composição do cenário de testes, foram utilizados dados de respiração obtidos a partir da base de dados Fantasia, do Physionet. Os registros de volume-minuto respiratório – quantidade total de ar que entra nas vias respiratórias a cada minuto – foram gerados e transferidos para o servidor MQTT utilizando o esquema de dados HDash.

O esquema de dados HDash permite modelar os esquemas de dados que poderão ser recebidos pelo servidor MQTT. Cada esquema possui uma URI (*Universal Resource Identifier*) que permite identificá-la de forma única, além de um código para versionamento. Assim, o Hash permite modelar diferentes esquemas de dados que poderão ser enviados para a aplicação eHealth, além de fornecer meios de validação dos dados recebidos pela aplicação, em conformidade com a estrutura previamente concebida.

Para executar os testes, o **Gerente de gateway** foi instanciado em um *cluster* de contêineres e foram observados: (i) consumo de memória, (i) consumo de CPU, (iii) rede

I/O, (iv) disco I/O e (v) pacotes de rede, em função do número de *gateways* instanciados. Além disso, no servidor MQTT foram monitorados: (vi) quantidade de clientes conectados, (vii) total de pacotes recebidos, (viii) total de *bytes* trafegados. Os testes foram executados simulando 100, 300 e 500 *gateways* conectados simultaneamente, enviando um registro a cada 10 segundos.

Os procedimentos necessários para a execução dos testes foram divididos em 3 etapas. Na primeira etapa, foi criado e configurado o *cluster* para o gerenciamento dos contêineres. A seguir, foram inicializadas 2 instâncias do MQTT: uma para o gerenciamento da comunicação entre o **Gerente de configuração** e o **Gerente de gateway** e outra para a recepção dos dados simulados. Na segunda etapa, foram instanciados os **Gerentes de gateway**, de acordo com a quantidade preestabelecida para aquela bateria de testes. Por fim, foram realizadas as configurações do HealthSimulator, utilizando a *interface Web*. Na etapa final, a simulação foi executada e os dados coletados.

Etapa1. Para criar o *cluster* de simulação, foram utilizados 3 máquinas com as seguintes configurações: um servidor ML310e Gen8 v2 com 8GB de memória, processador XEON 2 e 300GB de armazenamento e dois computadores com 24GB de memória, 240GB de armazenamento SSD e processador Intel Core i5 de terceira geração, ilustrado na Figura 35(2). Em cada servidor físico, foi realizada a instalação do Proxmox, um ambiente de virtualização de servidores de código aberto, cuja *interface* de gerenciamento é exemplificada na Figura 35(1). Para o gerenciamento dos contêineres foi configurado um *cluster* com Kubernetes, um sistema de orquestração de contêineres *open-source* para automatizar as tarefas de implantação, dimensionamento e gestão de aplicações baseadas em Docker, como é o caso do HealthSimulator. A gestão do Kubernetes via *interface* gráfica foi possível por meio da instalação do Rancher.

Em virtude da quantidade de mensagens a serem enviadas por meio do servidor MQTT durante a simulação, foi preciso empregar um servidor elástico que suportasse milhares de conexões simultâneas. Para alcançar este objetivo, foi escolhido o EMQX, um *broker* MQTT distribuído, de código aberto e escalável. A instalação do EMQX foi realizada por intermédio de uma imagem Docker, no ambiente Kubernetes. Para que fosse possível suportar a quantidade de conexões simultâneas propostas no teste, foi necessário realizar ajustes nos parâmetros do *Kernel*, conforme orientações disponibilizadas na página do projeto. Foi empregado o modo 'cluster automático' por meio do Kubernetes, utilizando 5 contêineres para cada instância do servidor.

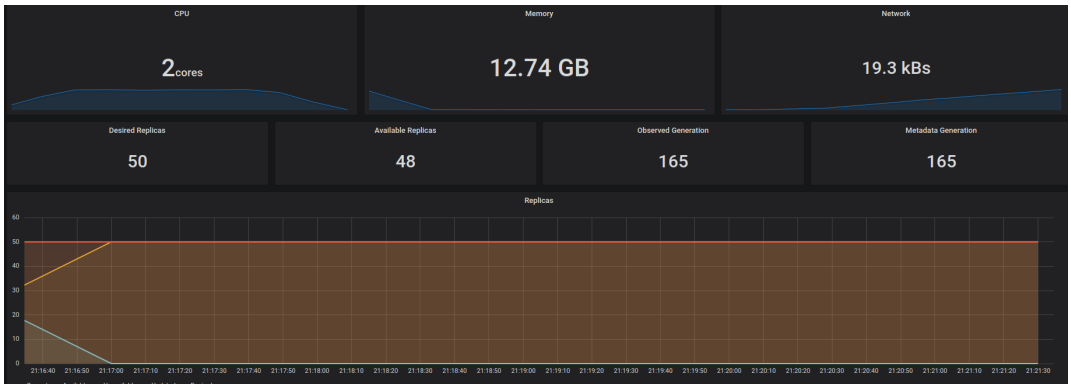
Duas ferramentas adicionais foram habilitadas no Rancher, um gerenciador de infraestrutura Docker com o propósito de coletar e exibir as informações de monitoramento durante a fase de execução dos testes. Para a coleta das informações foi utilizado o Prometheus, sistema de monitoramento para serviços e aplicações que armazena métricas em determinados intervalos de tempo. Por meio do Prometheus é possível ainda consultar

Figura 35 – Configuração do servidor

Fonte: Elaborado pelo autor

as informações coletadas, utilizando a linguagem PromQL, o que facilita a compilação dos resultados obtidos. Para o acompanhamento em tempo real da simulação, foi utilizado o Grafana, uma plataforma de observação que pode consultar os dados armazenados no Prometheus e exibi-los em *dashboards* interativos. A *interface* de gerenciamento da ferramenta Grafana, visualizada na Figura 36, pode ser personalizada por meio de parâmetros a fim de proporcionar um acompanhamento otimizado da infraestrutura durante a simulação.

Figura 36 – Interface de visualização da ferramenta Grafana



Fonte: Elaborado pelo autor

Etapa2. Na segunda etapa, foram instanciados os *gateways* para iniciar a simulação. Durante a bateria de testes esta etapa foi repetida três vezes para cada configuração

proposta (100, 300 e 500 *gateways* simulados). No que diz respeito à configuração, uma vez que cada instância do *gateway* consome aproximadamente 25MB de memória, optou-se por adicionar esta restrição à inicialização dos contêineres, prevenindo possíveis problemas de indisponibilidade de memória nas máquinas físicas.

Esta etapa foi especialmente desafiadora, em face dos problemas enfrentados acerca do consumo de memória do simulador, adversidade que foi sanada com a troca de algumas bibliotecas utilizadas em sua construção, como o Pandas, e com a otimização no envio dos dados do **Gerente de configuração** para o **Gerente de gateway**.

Etapa3. Na última etapa do processo, a simulação foi configurada por meio da *interface* principal do simulador. Os parâmetros de entrada foram preenchidos e a simulação executada por 5 minutos. A Tabela 2 sumariza a média totais das informações coletadas no MQTT durante a simulação:

Tabela 2 – Métricas coletadas no MQTT durante a execução da simulação

(**Clientes**)	(**Mensagens transmitidas**)	(**Bytes trafegados**)
100	3000	120000
300	8988	359520
500	14489	579560

Fonte: Elaborado pelo autor

A Figura 37 exibe os gráficos dos dados coletados sobre consumo de memória, consumo de CPU, rede I/O, disco I/O e pacotes de rede durante a execução da simulação, seguindo a mesma distribuição da Tabela 2. É possível observar um pico de utilização de disco e CPU durante a inicialização dos contêineres nos três cenários propostos. Como a quantidade de memória disponível para cada contêiner foi limitada, este parâmetro se manteve estável durante toda a simulação após a inicialização.

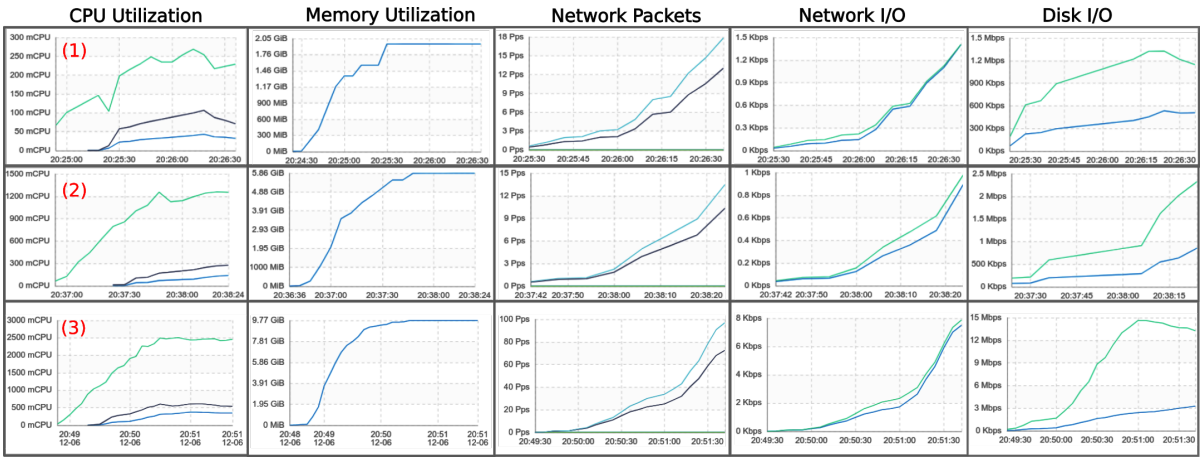


Figura 37 – Recursos monitorados durante a simulação.

As tentativas de escalar o simulador para além de 500 *gateways* conectados simultaneamente levaram a problemas de comunicação com o *broker* MQTT. Observou-se que, apesar das instâncias do simulador inicializarem com sucesso, a comunicação com o *broker* MQTT permanecia instável. Não foi possível determinar com precisão o motivo deste comportamento, uma vez que as otimizações realizadas na configuração do MQTT deveriam resultar no suporte às centenas ou milhares de conexões simultâneas. Como a simulação foi realizada sobre uma infraestrutura não profissional, é possível que a limitação esteja ligada aos recursos físicos disponíveis para consumo.

Apesar das questões apontadas anteriormente, o experimento realizado evidencia o potencial do HealthSimulator como ferramenta escalável para simulação de dados biomédicos, já que foi possível instanciar novos *gateways* apesar da limitação com o MQTT. Testes preliminares executados em um ambiente profissional culminaram na instanciação de 5 mil *gateways* simultaneamente. Todavia, será necessário executar novos testes neste ambiente a fim de estabelecer a causa do comportamento observado durante o experimento. Tais testes devem ainda determinar se o emprego de outro mecanismo para troca de mensagens entre o **Gerente de configuração** e o **Gerente de gateway** pode levar a melhores resultados.

5.3 Considerações finais do capítulo

Com a realização desta prova de conceito foi possível demonstrar o funcionamento da infraestrutura concebida e atestar que os requisitos estabelecidos no projeto da arquitetura proposta, particularmente os relacionados à comunicação com bases de dados biomédicos online e à escalabilidade, foram atendidos. Por meio dos testes realizados comprovou-se o potencial de uso do HealthSimulator, ficando evidente a adequação das decisões arquiteturais e as escolhas tecnológicas feitas no projeto.

O próximo capítulo encerra este documento, apresentando as conclusões finais sobre o trabalho desenvolvido e as perspectivas de pesquisas futuras.

6 Conclusões

Como destacado ao longo do texto, o aumento do número de pessoas idosas e com condições médicas crônicas, bem como das internações recorrentes destes pacientes, aliado à escassez de médicos qualificados e de leitos hospitalares, impulsionaram a demanda por serviços de monitoramento remoto de pacientes em suas próprias residências em diversos cantos do mundo.

Embora muitas soluções de monitoramento remoto tenham sido desenvolvidas, a maioria delas surgiu sem preocupações com a heterogeneidade do hardware de monitoramento - fortemente caracterizado pelo uso de dispositivos de IoT - com o uso ainda limitado de padrões de formatação e comunicação de dados, com os problemas de escalabilidade da plataforma de simulação que surgem à medida do aumento descontrolado do número de pacientes conectados simultaneamente à plataforma de monitoramento, e ao baixo suporte à manipulação de bases de dados biomédicos reais coletados de diferentes repositórios de dados de pesquisa médicas disponíveis.

A revisão da literatura realizada durante esta pesquisa comprovou que um problema atual na área de Informática em Saúde é a carência de ambientes de suporte para prototipagem e teste de serviços, componentes e aplicações em larga escala, onde se enquadram as aplicações de monitoramento de pacientes em domicílio baseada no uso de dispositivos IoT.

Ancorado neste *gap* tecnológico este trabalho apresentou o HealthSimulator, um ambiente de simulação para o cenário de monitoramento remoto de pacientes. O HealthSimulator facilita a avaliação de aplicações de Saúde ainda em fase inicial de desenvolvimento, tornando dispensável ambientar um cenário real de monitoramento o que, como já discutido, pode ser inviável tanto pelos altos custos quanto por impedimentos operacionais ou legais. Assim, usando o HealthSimulator o usuário pode realizar experimentos e testes que permitem antecipar respostas ou indicar problemas de implementação ou operação da sua aplicação de Saúde. De fato, quando uma alteração frequente da configuração não é possível no sistema real, torna-se muito difícil prever e controlar diferentes tipos de comportamentos de falha e, por conseguinte, os resultados alcançados podem não ser os esperados.

A versão atual do HealthSimulator fornece aos desenvolvedores das aplicações dados reais provenientes de repositórios de dados online abertos, usando padrões de informação conhecidos e referenciados pela comunidade de Saúde, bem como protocolos de comunicação padronizados para o acesso aos registros biomédicos dos pacientes. Oferece, também, uma interface gráfica de configuração e visualização, que permite aos desenvolvedores

acompanhar a execução das inferências realizadas pelo simulador de forma simples. Apesar de existirem simuladores de dados para IoT que possibilitem a interação e a percepção do mundo real, não há - no nosso conhecimento - e tendo por base a revisão de literatura realizada, uma ferramenta de simulação que seja independente de plataforma, escalável, e que permita a importação de dados bases importantes de Saúde disponíveis online, tais como PhysioNet e Kaggle. Testes iniciais mostraram a adequação da arquitetura conceitual e das escolhas tecnológicas usadas na implementação dos componentes do simulador proposto.

Em que pese as suas vantagens a implementação atual do HealthSimulator possui algumas limitações, vislumbrando-se, portanto, algumas possibilidade de trabalhos futuros. Dentre elas, pode-se citar a restrição de acesso a apenas um repositório de dados online, o Physionet. No entanto, a arquitetura do HealthSimulator já está preparada para a incorporação de novos repositórios, o que objetiva-se fazer na sequência desta pesquisa. Uma outra limitação diz respeito à necessidade de implementação de novos padrões de Saúde, tendo em vista que a versão atual do simulador implementa somente o padrão HL7 FHIR, em que pese a sua importância na comunidade de Informática em Saúde. Por fim, como apenas uma prova de conceito foi considerada nesta dissertação, é necessário aumentar o grau de complexidade de avaliação da infraestrutura proposta realizando experimentos e estudos de caso mais elaborados, preferencialmente com aplicações de monitoramento reais consumindo dados do simulador.

Referências

ABUKHOUSA, E.; MOHAMED, N.; AL-JAROODI, J. e-Health Cloud: Opportunities and Challenges. *Future Internet*, v. 4, n. 3, p. 621–645, 2012. Citado na página 49.

AGAR, B.; EREN, M.; CINAR, A. Glucosim: educational software for virtual experiments with patients with type 1 diabetes. p. 845–848, 2006. Citado 8 vezes nas páginas 3, 29, 31, 32, 33, 34, 40 e 43.

ALHOMSI, Y. et al. Real-Time Communication Network Using Firebase Cloud IoT Platform for ECMO Simulation. *Proceedings - 2017 IEEE International Conference on Internet of Things, IEEE Green Computing and Communications, IEEE Cyber, Physical and Social Computing, IEEE Smart Data, iThings-GreenCom-CPSCom-SmartData 2017*, v. 2018-Janua, p. 178–182, 2018. Citado na página 19.

BACHLER, M. et al. Simulation of physiologic ectopic beats in heartbeat intervals to validate algorithms. *IFAC-PapersOnLine*, Elsevier Ltd., v. 28, n. 1, p. 123–128, 2015. ISSN 24058963. Disponível em: <http://dx.doi.org/10.1016/j.ifacol.2015.05.105>. Citado na página 3.

BEN-KIKI, O.; EVANS, C.; INGERSON, B. YAML Ain't Markup Language (YAML™) Version 1.2. *Language*, p. 1–100, 2009. Disponível em: <http://www.yaml.org/spec/1.2/spec.html>. Citado na página 19.

BENDER, D.; SARTIPI, K. HL7 FHIR: An Agile and RESTful approach to healthcare information exchange. *Proceedings - IEEE Symposium on Computer-Based Medical Systems*, p. 326–331, 2013. ISSN 10637125. Citado 2 vezes nas páginas 3 e 16.

BERHANU, Y.; ABIE, H.; HAMDI, M. A testbed for adaptive security for IoT in eHealth. *Proceedings of the International Workshop on Adaptive Security - ASPI '13*, ACM Press, New York, New York, USA, n. 0314, p. 1–8, 2013. Disponível em: <http://dl.acm.org/citation.cfm?doid=2523501.2523506>. Citado na página 3.

BHOJWANI, S. N. Simulation of Physiological Signals using Wavelets. *Evaluation*, p. 73, 2007. Citado na página 20.

BLACKSTOCK, M.; LEA, R. Toward a Distributed Data Flow Platform for the Web of Things (Distributed Node-RED). 2014. Disponível em: <http://dx.doi.org/10.1145/2684432.2684439>. Citado na página 26.

BLEDA, A. L. et al. Evolution towards better AAL environments. *Proceedings - 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2013*, p. 518–523, 2013. Citado na página 2.

BOETTIGER, C. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, ACM, v. 49, n. 1, p. 71–79, jan 2015. ISSN 01635980. Disponível em: <http://dl.acm.org/citation.cfm?doid=2723872.2723882>. Citado 2 vezes nas páginas 21 e 22.

BRAY, A. et al. Pulse Physiology Engine: an Open-Source Software Platform for Computational Modeling of Human Medical Simulation. *SN Comprehensive Clinical Medicine*, SN Comprehensive Clinical Medicine, v. 1, n. 5, p. 362–377, 2019. Citado na página 3.

CELESTRINI, J. R. et al. HealthDash: Monitoramento remoto de pacientes utilizando programação baseada em fluxo de dados. In: *Anais do XIX Simpósio Brasileiro de Computação Aplicada à Saúde (SBCAS)*. Sociedade Brasileira de Computação - SBC, 2019. p. 222–233. Disponível em: <https://sol.sbc.org.br/index.php/sbcas/article/view/6256>. Citado na página 2.

CIUCU, R.-i.; METHODS, A. L. ECG Genertion Methods for Testing and a Maintenance of Cardiac Monittors. 2015. Citado 3 vezes nas páginas 3, 12 e 42.

COMMUNITY, W. *HTML Standard*. 2015. 1154 p. Disponível em: <https://html.spec.whatwg.org/multipage/web-sockets.html>. Citado na página 24.

COUTINHO, A. et al. Fogbed: A Rapid-Prototyping Emulation Environment for Fog Computing. *IEEE International Conference on Communications*, v. 2018-May, 2018. ISSN 15503607. Citado na página 20.

COUTO, F. A Framework for Mixed-Reality Simulations of Smart-Spaces. 2013. Citado na página 35.

CRISTINA, E. et al. REPOSITÓRIO DE DADOS CIENTÍFICOS : aspectos sobre privacidade de dados Scientific Data Repositories : aspects about data Privacy 1 Introdução Os Repositórios de dados científicos são ambientes implementados nas universidades com infraestrutura para dar su. v. 4, 2017. Disponível em: <http://150.162.242.35/handle/123456789/180294>. Citado na página 11.

D'ANGELO, G.; FERRETTI, S.; GHINI, V. Modeling the internet of things: A simulation perspective. *Proceedings - 2017 International Conference on High Performance Computing and Simulation, HPCS 2017*, p. 18–27, 2017. Citado na página 16.

DASTJERDI, A. V.; BUYYA, R. Fog Computing: Helping the Internet of Things Realize Its Potential. *Computer*, IEEE, v. 49, n. 8, p. 112–116, 2016. ISSN 00189162. Citado na página 24.

DE, A.; DO, N.; FOR, R. A Pattern-based Testing Framework for IoT Ecosystems. 2018. Citado 3 vezes nas páginas 38, 39 e 42.

DEWEY, H. H.; DEVRIES, D. R. Case study in utilizing the Internet of Things as a PHM architecture for aerospace applications. *IEEE Aerospace Conference Proceedings*, IEEE, v. 2018-March, p. 1–15, 2018. ISSN 1095323X. Citado na página 2.

DINCULEANă, D.; CHENG, X. Vulnerabilities and Limitations of MQTT Protocol Used between IoT Devices. *Applied Sciences*, v. 9, n. 5, p. 848, 2019. ISSN 2076-3417. Disponível em: <http://www.mdpi.com/2076-3417/9/5/848>. Citado na página 23.

DIZDAREVIĆ, J. et al. A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration. *ACM Computing Surveys*, v. 51, n. 6, p. 1–29, 2019. ISSN 15577341. Citado na página 25.

Docker Inc. *What is a Container | Docker*. 2017. Disponível em: <https://www.docker.com/resources/what-containerhttps://www.docker.com/what-container>. Citado na página 22.

Docker Inc. *Enterprise Container Platform | Docker*. 2019. Disponível em: <https://www.docker.com/>. Citado na página 21.

EDELMANN, J. C. et al. An ECG simulator with a novel ECG profile for physiological signals. *Journal of Medical Engineering and Technology*, Taylor & Francis, v. 42, n. 7, p. 501–509, 2018. ISSN 1464522X. Disponível em: <https://doi.org/10.1080/03091902.2019.1576788>. Citado 2 vezes nas páginas 3 e 12.

ERZEN, F. C.; BIROL, G.; ÇINAR, A. Glucosim: A simulator for education on the dynamics of diabetes mellitus. *Annual Reports of the Research Reactor Institute, Kyoto University*, v. 4, p. 3163–3166, 2001. ISSN 04549244. Citado na página 32.

GOLDBERGER, A. L. et al. PhysioBank, PhysioToolkit, and PhysioNet. *Circulation*, Ovid Technologies (Wolters Kluwer Health), v. 101, n. 23, jun 2000. ISSN 0009-7322. Citado na página 11.

GSMA. Digital Healthcare Interoperability. n. October, p. 41, 2016. Disponível em: <https://www.gsma.com/iot/wp-content/uploads/2016/10/Interoperability-report-v1.2.pdfhttp://www.gsma.com/connectedliving/pwc-report-realising-the-benefits-of-mobile-iot-solutions/>. Citado 6 vezes nas páginas 14, 15, 18, 46, 47 e 49.

HOSSAIN, M. et al. IoTbed: A generic architecture for testbed as a service for internet of things-based systems. *Proceedings - 2017 IEEE 2nd International Congress on Internet of Things, ICIOT 2017*, p. 42–49, 2017. Citado 3 vezes nas páginas 11, 21 e 52.

Institute for Health Metrics and Evaluation. *Global spending on health is expected to increase to $18.28 trillion worldwide by 2040 but many countries will miss important health benchmarks*. 2016. Disponível em: <http://www.healthdata.org/news-release/global-spending-health-expected-increase-1828-trillion-worldwide-2040-many-countries>. Citado na página 1.

ISLAM, S. M. et al. The internet of things for health care: A comprehensive survey. *IEEE Access*, v. 3, p. 678–708, 2015. ISSN 21693536. Citado na página 1.

KARAGIANNIS, V. et al. Sensus: Smart Water Network. *Transaction on IoT and Cloud Computing*, v. 3, n. 1, p. 1–10, 2016. ISSN 2331-4761. Disponível em: <http://sensus.com/smart-water-network/>. Citado na página 19.

KAYLA, M. *6 Exciting IoT Use Cases in Healthcare*. 2018. Disponível em: <https://www.iotforall.com/exciting-iot-use-cases-in-healthcare/>. Citado na página 2.

KERTESZ, A.; PFLANZNER, T.; GYIMOTHY, T. A Mobile IoT Device Simulator for IoT-Fog-Cloud SystemsAracruz, Espírito Santo. *Journal of Grid Computing*, Journal of Grid Computing, n. June, 2018. ISSN 15729184. Citado 3 vezes nas páginas 29, 36 e 37.

KOKALKI, S. A. et al. Smart health band using IoT. *IEEE International Conference on Power, Control, Signals and Instrumentation Engineering, ICPCSI 2017*, IEEE, p. 1683–1687, 2018. Citado na página 11.

LATVAKOSKI, J. et al. *A Survey on M2M Service Networks*. [S.l.: s.n.], 2014. v. 3. 130–173 p. ISBN 3582072223. Citado na página 51.

LEKIĆ, M.; GARDAŠEVIĆ, G. IoT sensor integration to Node-RED platform. *2018 17th International Symposium on INFOTEH-JAHORINA, INFOTEH 2018 - Proceedings*, v. 2018-Janua, n. March, p. 1–5, 2018. Citado na página 26.

LENZ, R.; REICHERT, M. IT support for healthcare processes - premises, challenges, perspectives. *Data and Knowledge Engineering*, v. 61, n. 1, p. 39–58, 2007. ISSN 0169023X. Citado na página 14.

LIU, Y. et al. A Novel Cloud-Based Framework for the Elderly Healthcare Services Using Digital Twin. *IEEE Access*, IEEE, v. 7, p. 49088–49101, 2019. ISSN 21693536. Disponível em: <https://ieeexplore.ieee.org/document/8686260/>. Citado na página 14.

LO, O. Owen Lo Thesis. n. May, 2015. Citado na página 3.

LO, O. et al. Patient Simulator: Towards Testing and Validation of e-Health Infrastructures. *Proceedings of the Pervasive Health*, n. March 2014, 2011. Citado 6 vezes nas páginas 29, 30, 31, 39, 42 e 43.

LO, O. et al. Towards Simulation of Patient Data for Evaluation of E-Health Platform and Services. *13th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting*, n. December, 2012. Disponível em: <http://goo.gl/aVqeV>. Citado 2 vezes nas páginas 3 e 21.

LOOGA, V.; OU, Z.; DENG, Y. MAMMotH: a massive-scale emulation platform for internet of things. *International Conference on Cloud Computing and Intelligence Systems*, p. 1235, 2012. Citado 4 vezes nas páginas 29, 33, 40 e 43.

MAZZER, D.; FRIGIERI, E.; PARREIRA, L. Protocolos M2M para Ambientes Limitados no Contexto do IoT: Uma Comparação de Abordagens. *Inatel.Br*, 2015. Disponível em: <http://www.inatel.br/smartcampus/imgs/protocolos-para-iot-pt.pdf>. Citado na página 24.

MCCONALOGUE, E.; DAVIS, P.; CONNOLLY, R. The Role of Total Cost of Ownership Tools in AAL Technology Assessment. *SSRN Electronic Journal*, n. September, p. 425–429, 2018. Citado 2 vezes nas páginas 7 e 71.

MEMON, M. et al. Ambient Assisted Living healthcare frameworks, platforms, standards, and quality attributes. *Sensors (Switzerland)*, v. 14, n. 3, p. 4312–4341, 2014. ISSN 14248220. Citado na página 2.

Ministério da Saúde. Portaria nº 341, de 31 de agosto de 2011. *Diário Oficial da União*, Cartilha s, p. 81–90, 2011. Disponível em: <http://bvsms.saude.gov.br/bvs/saudelegis/gm/2011/prt2073{_}31{_}08{_}>. Citado na página 13.

MIYOSHI, N. S. B. *Arquitetura e métodos de integração de dados e interoperabilidade aplicados na saúde mental*. Tese (Doutorado) — Universidade de São Paulo, Ribeirão Preto, jul 2018. Disponível em: <http://www.teses.usp.br/teses/disponiveis/17/17138/tde-20072018-100724/>. Citado na página 13.

MODELING, E.; LO, O. Ph.D Year 1: Transfer Report (RD5) Towards a Patient Simulator Framework for Evaluation of e-Health Environments: Modeling, Techniques, Validation and Implementation. n. February, 2011. Citado 3 vezes nas páginas 30, 40 e 42.

MOODY, G. B.; MARK, R. G.; GOLDBERGER, A. L. Physionet: A web-based resource for the study of physiologic signals. *IEEE Engineering in Medicine and Biology Magazine*, v. 20, n. 3, p. 70–75, 2001. ISSN 07395175. Citado na página 12.

MOODY, G. B.; MARK, R. G.; GOLDBERGER, A. L. PhysioNet: Physiologic signals, time series and related open source software for basic, clinical, and applied research. *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, IEEE, p. 8327–8330, 2011. ISSN 1557170X. Citado na página 12.

MORRISON, J. P. *Flow-Based Programming, 2nd Edition: A New Approach to Application Development*. Scotts Valley, CA: CreateSpace, 2010. ISBN 1451542321. Citado na página 25.

MSHALI, H. et al. A survey on health monitoring systems for health smart homes. *International Journal of Industrial Ergonomics*, Elsevier B.V, v. 66, p. 26–56, 2018. ISSN 18728219. Disponível em: <https://doi.org/10.1016/j.ergon.2018.02.002>. Citado na página 8.

MUBASHIR, M.; SHAO, L.; SEED, L. A survey on fall detection: Principles and approaches. *Neurocomputing*, Elsevier, v. 100, p. 144–152, 2013. ISSN 09252312. Disponível em: <http://dx.doi.org/10.1016/j.neucom.2011.09.037>. Citado na página 11.

NAIK, N. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. *2017 IEEE International Symposium on Systems Engineering, ISSE 2017 - Proceedings*, 2017. Citado 2 vezes nas páginas 23 e 24.

OJIE, E.; PEREIRA, E. Simulation tools in internet of things. p. 1–7, 2018. Citado na página 20.

PAGLIARI, C. et al. *What is eHealth (4): A scoping exercise to map the field*. 2005. Citado na página 18.

PALATTELLA, M. R. et al. Internet of Things in the 5G Era: Enablers, Architecture, and Business Models. *IEEE Journal on Selected Areas in Communications*, IEEE, v. 34, n. 3, p. 510–527, 2016. ISSN 07338716. Citado na página 2.

PFLANZNER, T.; FIDRICH, M.; KERTESZ, A. Simulating Sensor Devices for Experimenting with IoT Cloud Systems. p. 105–126, 2017. Citado na página 36.

PFLANZNER, T. et al. MobIoTSim: Towards a mobile IoT device simulator. In: *Proceedings - 2016 4th International Conference on Future Internet of Things and Cloud Workshops, W-FiCloud 2016*. [S.l.]: IEEE, 2016. p. 21–27. ISBN 9781509039463. ISSN 00029440. Citado 4 vezes nas páginas 35, 36, 37 e 43.

PONTES, P. M.; LIMA, B.; FARIA, J. P. Test patterns for IoT. In: *Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation - A-TEST 2018*. New York, New York, USA: ACM Press, 2018. p. 63–66. ISBN 9781450360531. Disponível em: <http://dl.acm.org/citation.cfm?doid=3278186.3278196>. Citado 2 vezes nas páginas 21 e 38.

PONTES, P. M.; LIMA, B.; FARIA, J. P. Izinto : A Pattern-Based IoT Testing Framework. In: *Companion Proceedings for the ISSTA/ECOOP 2018 Workshops on - ISSTA '18*. New York, New York, USA: ACM Press, 2019. p. 125–131. ISBN 9781450359399. Disponível em: <http://dl.acm.org/citation.cfm?doid=3236454.3236511>. Citado 6 vezes nas páginas 29, 34, 37, 39, 41 e 43.

RAJA, S. P.; RAJKUMAR, T. D.; RAJ, V. P. Internet of Things: Challenges, Issues and Applications. *Journal of Circuits, Systems and Computers*, v. 27, n. 12, p. 1830007, 2018. ISSN 0218-1266. Citado na página 1.

REYNOLDS, V.; CAHILL, V.; SENART, A. Requirements for an ubiquitous computing simulation and emulation environment. p. 1, 2006. Citado na página 11.

RGHIOUI, A. et al. Internet of Things for Measuring Human Activities in Ambient Assisted Living and e-Health. *Network Protocols and Algorithms*, v. 8, n. 3, p. 15, 2017. Citado 3 vezes nas páginas 1, 7 e 8.

Ricardo Neubauer Moralles, C.; ALEGRE, P. Proposta de um Sistema para troca de informações odontológicas de pacientes entre instituições de saúde brasileiras com o sistema de informação em saúde canadense. 2018. Citado na página 18.

RODRIGUES, J. Strategic information systems: concepts, methodologies, tools, and applications. *Choice Reviews Online*, Medical Information Science Reference, v. 47, n. 08, p. 47–4186–47–4186, 2010. ISSN 0009-4978. Citado na página 14.

SANTOS, B. P. et al. Internet das Coisas: da Teoria à Prática. *Minicursos SBRC-Simp*, p. 50, 2016. Disponível em: <https://homepages.dcc.ufmg.br/{~}mmvieira/cc/papers/internet-das-coisas.p>. Citado na página 1.

SHOUMIK, F. S. et al. Scalable micro-service based approach to FHIR server with golang and No-SQL. *20th International Conference of Computer and Information Technology, ICCIT 2017*, v. 2018-Janua, p. 1–5, 2018. Citado na página 17.

SU, C. R. et al. A novel framework for a remote patient monitoring (RPM)system with abnormality detection. *Health Policy and Technology*, Elsevier Ltd, v. 8, n. 2, p. 157–170, 2019. ISSN 22118845. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S2211883718302569>. Citado 2 vezes nas páginas 2 e 7.

SULISTIO, A.; YEO, C. S.; BUYYA, R. A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. *Software - Practice and Experience*, v. 34, n. 7, p. 653–673, 2004. ISSN 00380644. Citado na página 39.

SURAKI, M. Y.; JAHANSHAHI, M. Internet of things and its benefits to improve service delivery in public health approach. *AICT 2013 - 7th International Conference on Application of Information and Communication Technologies, Conference Proceedings*, IEEE, p. 1–4, 2013. Citado na página 2.

TANTAWI, M. M. et al. An evaluation of the generalisability and applicability of the PhysioNet electrocardiogram (ECG) repository as test cases for ECG-based biometrics. *International Journal of Cognitive Biometrics*, v. 1, n. 1, p. 66, 2012. ISSN 2042-6461. Citado na página 12.

Tsiachri Renta, P.; SOTIRIADIS, S.; PETRAKIS, E. G. Healthcare Sensor Data Management on the Cloud. p. 25–30, 2017. Citado na página 2.

Ud Din, I. et al. The Internet of Things: A Review of Enabled Technologies and Future Challenges. *IEEE Access*, IEEE, v. 7, p. 7606–7640, 2019. ISSN 21693536. Citado na página 1.

VASE, T. Advantages of Docker. p. 1–24, 2015. Citado na página 21.

WALINJKAR, A. FHIR Tools for Healthcare Interoperability. *Biomedical Journal of Scientific & Technical Research*, v. 9, n. 5, p. 4–7, 2018. Citado na página 16.

ZAMANIFAR, A.; NAZEMI, E. An approach for predicting health status in IoT health care. *Journal of Network and Computer Applications*, Elsevier Ltd, v. 134, n. January, p. 100–113, 2019. ISSN 10958592. Disponível em: <https://doi.org/10.1016/j.jnca.2019.02.029>. Citado 2 vezes nas páginas 11 e 12.