UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS I
CENTRO CIÊNCIA E TECNOLOGIA
CURSO DE CIENCIA DA COMPUTAÇÃO

ALISSON SANTOS DE SOUZA

ESPECIFICAÇÃO DE UMA INTERFACE DE COMUNICAÇÃO ENTRE UM AGREGADOR DE DADOS EM SAÚDE E UM AMBIENTE CLINICO INTEGRADO

CAMPINA GRANDE - PB
2016

ALISSON SANTOS DE SOUZA

ESPECIFICAÇÃO DE UMA INTERFACE DE COMUNICAÇÃO ENTRE UM AGREGADOR DE DADOS EM SAÚDE E UM AMBIENTE CLINICO INTEGRADO

Trabalho de Conclusão de Curso em Ciência da Computação da Universidade Estadual da Paraíba, como requisito à obtenção do título de bacharel em Ciência da Computação

Área de concentração: Engenharia de software

Orientador: Prof. Dr. Paulo Eduardo e Silva Barbosa

CAMPINA GRANDE
2016

S729e Souza, Alisson Santos de.
Especificação de uma interface de comunicação entre um agregador de dados em saúde e um ambiente clínico integrado [manuscrito] / Alisson Santos de Souza. - 2016.
57 p. : il. color.

Digitado.
Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia, 2016.
"Orientação: Prof. Dr. Paulo Eduardo e Silva Barbosa, Departamento de Computação".

1. Monitoramento. 2. Conectividade. 3. Engenharia de software. 4. Interface de comunicação. I. Título.
21. ed. CDD 005.1

ALISSON SANTOS DE SOUZA

ESPECIFICAÇÃO DE UMA INTERFACE DE COMUNICAÇÃO ENTRE UM AGREGADOR DE DADOS EM SAÚDE E UM AMBIENTE CLINICO INTEGRADO

Trabalho de Conclusão de Curso de Graduação em Ciência da Computação da Universidade Estadual da Paraíba, como requisito à obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 01 de Dezembro de 2016.

Prof. Dr. Paulo Eduardo e Silva Barbosa (UEPB)
Orientado (a)

Prof. MSc. Fábio Luiz Leite Júnior (UEPB)
Examinador(a)

Prof. Dr. Kezia de Vasconcelos Oliveira Dantas (UEPB - Campus VII)
Examinador(a)

A minha família e amigos, pela dedicação, companheirismo e amizade DEDICO.

AGRADECIMENTOS

Inicialmente gostaria de agradecer à minha família por todo o apoio e cuidados para que eu pudesse chegar ao fim dessa graduação, especialmente a minha mãe por todo seu esforço e dedicação em me manter focado nos estudos.

Ao professor Paulo Eduardo e Silva Barbosa, pela orientação, amizade, por ter me iniciado nessa linha de pesquisa que teve como fruto esse trabalho de conclusão de curso. Vi no professor Paulo uma pessoa que demonstra cuidados aos seus orientandos, uma qualidade que admiro muito em um professor.

A Lucas Barbosa, meu amigo e colega de laboratório que me ajudou no entendimento do D-Bus, uma ferramenta que faz parte desse trabalho.

A todos os professores que fizeram parte da minha formação, sem eles nada disso seria possível.

Aos meus colegas de sala, pela amizade, bons momentos que vivenciei com eles, e todas as experiências trocadas. E também pela compreensão nesse último período de curso, onde estive muito ausente.

E aos demais que contribuíram de forma direta, ou indireta para minha formação profissional e pessoal.

RESUMO

Ao analisarmos os atuais ambientes clínicos podemos observar vários dispositivos hospitalares trabalhando de forma independente, fragmentada, sem a capacidade de colaborarem entre si, perdendo assim a oportunidade de construir ambientes mais completos voltados aos pacientes. Por exemplo, poderíamos agrupar dados de diferentes dispositivos para diagnosticar doenças através de métodos de seleção de dados antes mesmo que elas se manifestem, ou até mesmo usar dados de dispositivos médicos para entender a dispersão de uma epidemia, entre outras inúmeras aplicações. Os ambientes clínicos integrados (do inglês *Integrated Clinicals Environments*) (ICE's) são uma aplicação prática da internet das coisas voltada para ambientes clínicos, tais ambientes são sistemas consumidores de dados que se comunicam com dispositivos médicos hospitalares. Esses sistemas buscam reduzir a fragmentação existente no atual modelo de ambiente clinico através de interoperabilidade, além de centralizar os dados de diversos dispositivos em um único local. Por conta disso, são vistos como solução para o atual panorama organizacional de ambientes clínicos. Percebendo o potencial dos ICE's, observou-se neles uma oportunidade de aprendizado e contribuição. A procura de entender melhor o funcionamento desse tipo de tecnologia foi-se necessário estudar uma implementação *open source* de um ICE, chamada OpenICE. Após compreender seu funcionamento se viu abertura para maximizar o poder de monitoramento que essa ferramenta tem por meio de uma integração com um sistema chamado Health Aggregator Manager (HAM). O HAM assim como o OpenICE é um sistema de monitoramento, contudo com um diferencial. Que é sua capacidade de se comunicar com dispositivos médicos pessoais, que por padrão são dispositivos de menor porte, que a população em geral tem mais acesso. Com a integração do HAM com o OpenICE, temos um ICE que passa a monitorar não apenas dispositivos hospitalares, mas também dispositivos pessoais, abrindo assim o leque de opções e aplicações de um ICE no mundo real. Para que a comunicação entre os sistemas fosse possível, se fez necessário o desenvolvimento de um adaptador que transmite os dados recolhidos pelo HAM para o OpenICE através de um middleware chamado Data Distribution Service (DDS). Tendo em vista a construção dessa solução integradora, esse trabalho busca especificar os passos e as tecnologias necessárias para a elaboração de uma interface de comunicação entre esses dois sistemas.

Palavras-Chave: monitoramento, conectividade, internet das coisas

ABSTRACT

When we analyzed the current clinical environments we could see several mdeical devices working independently, fragmented, without the capacity to collaborate, thus losing the opportunity to build more complete environments geared to patients. For example, we could group data from different devices to diagnose diseases through data selection methods even before they arise, or even use data from medical devices to understand the spread of an epidemic, among numerous other applications. The Integrated Clinical Environments (ICE's) - have as its basic operation monitor hospital medical devices - seek to reduce the fragmentation in the current clinical environment model, and centralizes data from multiple devices in a single location. Because of this, they are seen as a solution to the current organizational landscape of clinical environments. Realizing the potential of ICE's, we observed them an opportunity for learning and contribution. Seeking to better understand the functioning of this type of technology was a need to study an open source implementation of an ICE called OpenICE. Once we understand its operation is seen opening to maximize the power of monitoring that this tool has through integration with a system called Health Aggregator Manager (HAM), which like the OpenICE is a monitoring system, but with a difference, that is its ability to communicate with personal medical devices, which by nature are smaller devices that the general population has more access. With this integration the OpenICE starts to monitor not only hospital devices, but also personal devices, thus opening up the range of options and applications of ICE in the real world. For communication between the systems was possible, the development of an adapter that transmits the data collected by HAM to the OpenICE through a middleware called Data Distribution Service (DDS). In view of the construction of this integrated solution, this paper seeks to specify the steps and technologies necessary for the development of an interface between these two systems.

Keywords: monitoring, connectivity, Internet of things

LISTA DE ILUSTRAÇÕES

Figura 1 - Interface do HAM 13
Figura 2 - Arquitetura que envolve o HAM 14
Figura 3 - Arquitetura geral OpenICE 16
Figura 4 - Lista de dispositivos que podem ser simulados 17
Figura 5 - Monitor multiparametrico simulado 18
Figura 6 - Interface do supervisor do OpenICE 19
Figura 7 - Janela da ferramenta Alarm History 20
Figura 8 - Janela da ferramenta Auto Validate 21
Figura 9 - Janela da ferramenta Patient ID 22
Figura 10 - Monitor multiparametrico isolado no contexto de um paciente 23
Figura 11 - Camadas de uma aplicação DDS 24
Figura 12 - Entidades do DDS 25
Figura 13 - a) Uma rede de computadores padrão. b) A mesma rede separada por Domains 26
Figura 14 - OpenICE enquanto um hub de HAM's 30
Figura 15 - a) Representação das mensagens do HAM. b) Representação das mensagens do OpenICE 31
Figura 16 - *Topic* específico para o HAM 32
Figura 17 - Funcionamento do D-Bus no Adaptador DDS 33
Figura 18 - Diagrama estrutural da comunicação D-Bus da interface 34
Figura 19 - Topic DeviceIdentity em código 35
Figura 20 - Topic DeviceConnectivity em código 35
Figura 21 - Diagrama estrutural da fase de descoberta 36
Figura 22 - Autodescoberta do OpenICE 36
Figura 23 - OpenICE encontrando o HAM 37
Figura 24 - Funcionamento do adaptador DDS 38
Figura 25 - Janela do simulador de oxímetro 39
Figura 26 - Janela de monitoramento do HAM 40
Figura 27 - Recebimento de medição pelo OpenICE 41

LISTA DE ABREVIATURAS E SIGLAS

BPM	Batimento por minuto
DDS	Data Distribution Service
HAM	Health Aggregator Manager
ICE	Integrated Clinical Envorinment
IDL	Interactive Data Language
MD	Medical Device
NUTES	Núcleo de Tecnologias Estratégicas em Saúde
PnP	Plug and Play
RTI	Real-Time Inovations
SpO2	Saturação de oxigênio no sangue
UEPB	Universidade Estadual da Paraíba

SUMÁRIO

1. INTRODUÇÃO 10
2. REFERÊNCIAL TEÓRICO 12
2.1 Health Aggregator Manager (HAM) 12
2.1.1 Antidote 14
2.1.2 Continua Health Alliance 15
2.4 OpenICE 15
2.4.1 DDS 23
2.4.2 Entidades do DDS 25
2.4.3 Connext DDS 27
2.4.3.1 A ferramenta Code Generator e a IDL 27
2.6 D-Bus 27
2.7 Engenharia reversa em softwares 28
3. ENTENDENDO A INTERFACE DE COMUNICAÇÃO 29
3.1 A interface de comunicação 29
3.2 A construção do adaptador DDS 30
3.3 Estabelecendo a comunicação HAM-Adaptador 37
3. AVALIANDO A SOLUÇÃO 39
4.1 A exibição dos dados. 39
4.2 Validando através de testes 42
5. CONCLUSÃO 43
REFERÊNCIAS 45
APÊNDICE A – IDL do HAM 46
APÊNDICE B – Testes realizados 47

1. INTRODUÇÃO

Com a evolução da tecnologia da informação (TI) muitas áreas do conhecimento vêm se modernizando como, por exemplo, a engenharia, e a medicina. A medicina por sua vez, hoje é uma das principais dependentes da tecnologia informatizada, visando potencializar os cuidados aos seus pacientes, e principalmente aumentar as chances de salvar vidas. A tecnologia da informação está se tornando algo cada vez mais intrínseca à área de saúde, e uma das tecnologias que está chamando uma atenção especial para si é a saúde conectada.

Vista como um campo de inovação, a saúde conectada vem sendo discutida cada vez mais no meio empresarial e acadêmico por instituições como a MD PnP[1] e a CIMIT[2]. O objetivo dessa tecnologia é transformar ambientes que possuem dispositivos de cuidados com a saúde, que por padrão funcionam de forma heterogênea, sem saber da existência de outros dispositivos, em ambientes conectados onde todos os dispositivos trabalham em conjunto visando uma maximização do monitoramento do paciente. Além de existir a possibilidade de fornecer os dados dos dispositivos na rede, permitindo assim o monitoramento remoto de pacientes. Aplicações práticas de tais ambientes conectados podemos encontrar em estudos como Medical Device Connectivity for Improving Safety and Efficiency[3] (GOLDMAN, 2006), que idealiza a aplicação de ambientes conectados para coordenar bombas de infusão. Já aplicação de propósito geral como diagnostico, terapias, e monitoramento são melhores descritas em HITSP Device Connectivity Technical Note[4] (HITSP, 2010).

Entre os benefícios trazidos por essa tecnologia temos a agilidade na tomada de decisão, minimização do erro humano, e redução de custos o que a torna extremamente atraente para os profissionais de saúde. A implementação desse tipo de tecnologia pode variar, de forma que podemos encontrar dispositivos médicos pessoais, como aferidores de pressão, balanças, termômetros, e dispositivos médicos hospitalares, como monitores multiparemetricos, bombas de infusão entre outros. Ambos os tipos de dispositivos construídos pensando na saúde conectada,

[1] http://www.mdpnp.org/
[2] https://www.cimit.org
[3] http://www.mdpnp.org/uploads/ASA_May_2006_Newsletter_on_MD_PnP.pdf
[4] http://www.hitsp.org/Handlers/HitspFileServer.aspx?FileGuid=841f96d4-2c4f-47e5-aab5-7721902aff0b

para diversos públicos-alvo. Apesar de parecerem mundos diferentes, esses dispositivos podem se relacionar através de um conceito que chamamos de interoperabilidade. Segundo (HIMSS, 2013), na área da saúde, a interoperabilidade é a capacidade de diferentes sistemas de tecnologia da informação e aplicações de software se comunicarem, trocarem dados, e fazerem uso das informações que foram trocadas. Em outras palavras, existe um habitat onde todos esses dispositivos podem “conviver”, mas ele só é acessível através da interoperabilidade.

Com esse contexto em mente, esse trabalho terá como principal objetivo descrever uma interface de comunicação entre dois ambientes de saúde conectada, garantindo assim a interoperabilidade entre sistemas. Um ambiente proporcionado pelo *Health Aggregator Manager* (HAM) que se trata de um sistema voltado para monitoramento de dispositivos médicos pessoais, que funciona primordialmente por meio da tecnologia de comunicação *bluetooth*. E o outro pelo OpenICE, que de acordo com (MDPNP, 2014) pode ser descrito como um framework de saúde voltado para dispositivos médicos hospitalares, que utiliza a rede local como meio de transmissão de dados. Os dois sistemas podem ser considerados consumidores de dados e exercem funções semelhantes. Contudo, devido à suíte de aplicativos disponibilizada pelo OpenICE ser de grande valia para atividades de pesquisas envolvendo ambientes conectados, tornando-o o principal sistema dessa interface. Além disso, propomos uma estrutura em que os dados destinados a ambos os sistemas são monitorados pelo OpenICE.

1.1. Objetivos Gerais

Especificar e desenvolver uma interface de comunicação entre o HAM e o OpenICE, com a finalidade de aumentar a capacidade de monitoramento do OpenICE, tornando-o uma solução ainda mais completa para ambientes conectados.

1.2. Objetivos Específicos

- Obter dados provenientes de dispositivos médicos pessoais através do HAM.
- Transferir os dados do HAM para o OpenICE utilizando a tecnologia DDS.
- Exibir os dados que foram capturados pelo OpenICE provenientes do HAM.
- Documentar o processo de adaptação do HAM ao OpenICE.

2. REFERÊNCIAL TEÓRICO

Nessa seção será demonstrado as tecnologias, e técnicas necessárias para a construção da interface de comunicação proposta nesse trabalho. Inicialmente, iremos descrever o HAM, e posteriormente o OpenICE. Além disso, iremos citar quais tecnologias foram utilizadas para que fosse viável o desenvolvimento da interface de comunicação.

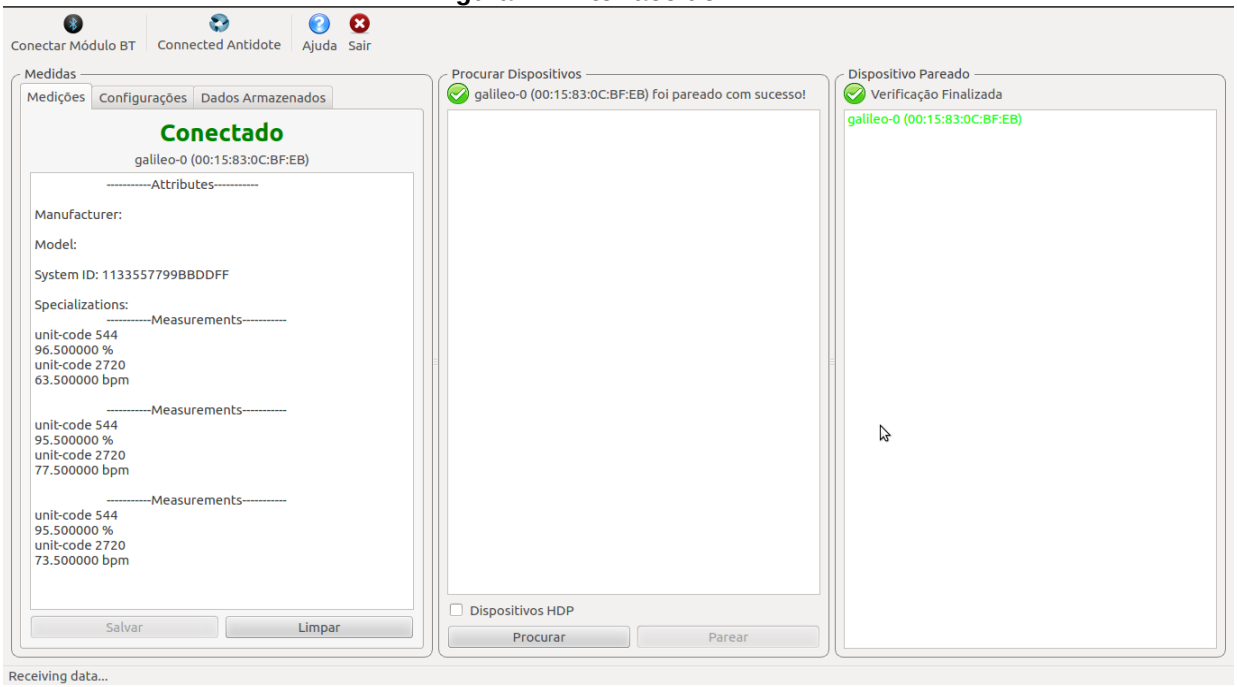
2.1 Health Aggregator Manager (HAM)

O HAM[5] é um software que foi desenvolvido nos laboratórios do NUTES, em parceria com uma empresa especialista em saúde conectada a Signove. O HAM é para o NUTES uma das suas principais contribuições para o universo *open source .*

O HAM se encaixa em uma categoria de software chamada “Agregador de dados em saúde”, e com ele temos a capacidade de recolher dados oriundos de dispositivos médicos pessoais como por exemplo: aferidor de pressão, glicosímetro, balança, oxímetro entre outros. Tudo isto é feito via bluetooth ou USB, sendo possível salvar os dados temporariamente ou permanentemente. Assim que os dados são recebidos pelo HAM, eles são exibidos como pode ser visto na Figura 1, tal visualização de dados é fruto de um processamento, já que por padrão o HAM constrói os dados utilizando arquivos XML. Contudo, existe também a opção de visualizar os arquivos XML construídos pelo HAM. Essa função tem propósitos mais técnicos, como depuração por exemplo.

[5] nutes.uepb.edu.br/ham

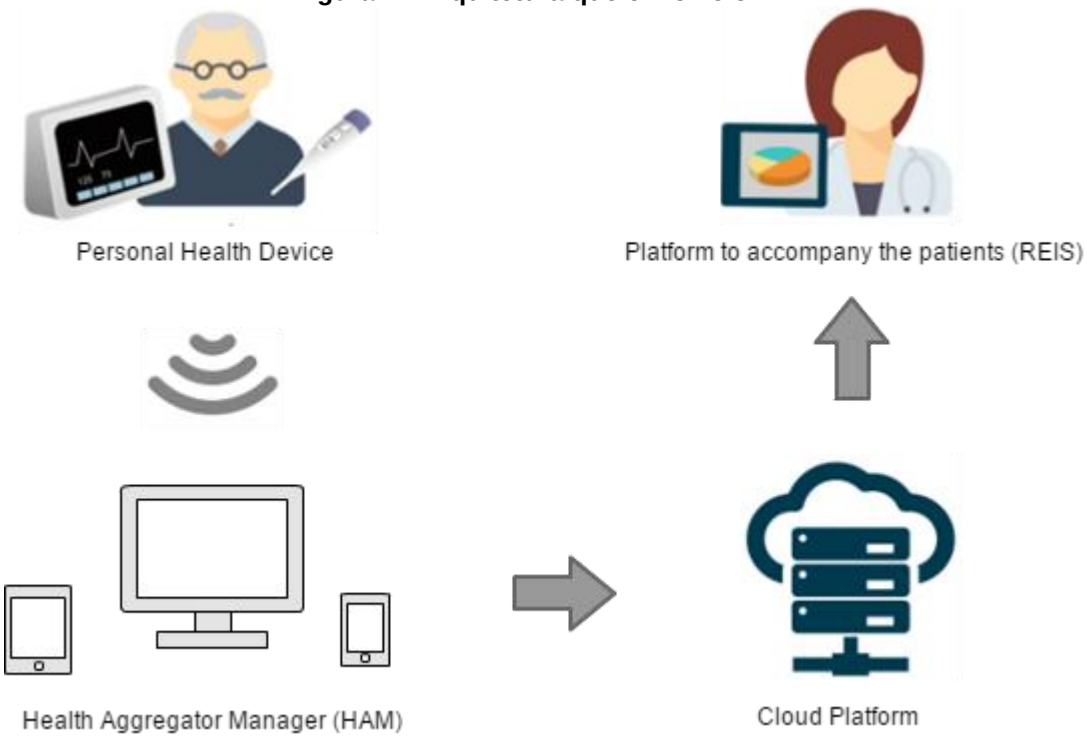
Figura 1 - Interface do HAM



Fonte: Elaborada pelo autor

Considerando o potencial do HAM de recolher e manipular dados, podemos usá-lo para por exemplo, enviar dados relacionados a um determinado paciente para um prontuário online onde profissionais de saúde como médicos e enfermeiros podem remotamente visualizar essas informações, analisa-las, e adiantar diagnósticos antes mesmo de entrarem em contato com o paciente, agilizando assim o atendimento. Além disso, podemos utilizar os dados armazenados pelo HAM de modo que seja possível criar um histórico daquele paciente para futuros rastreamentos de enfermidades que venham a surgir. Essas possibilidades existem graças a um segundo sistema também desenvolvido no NUTES, o REIS, no entanto ele foge do escopo desse trabalho, então não será necessário aborda-lo. Na Figura 2 temos uma ilustração do funcionamento do ecossistema do HAM.

Figura 2 - Arquitetura que envolve o HAM



Fonte: Site da Signove[6]

2.1.1 Antidote

Um grande diferencial do HAM está no modo em que ele se comunica com outros dispositivos, pois se baseia na norma ISO/IEEE 11073. Ela descreve uma biblioteca especializada em comunicação via *bluetooth*/USB de dispositivos médicos com computadores. De acordo com (JÁCOME, 2010), uma biblioteca pode ser descrita como componentes de software pré-escritos por outros programadores que resolve determinados problemas sem que você precise “reinventar a roda”, ou seja, sem que você precise escrever código já elaborado. Para facilitar a implementação da ISO/IEEE 11073 no HAM, foi utilizada a biblioteca *Antidote*, que é a implementação dessa norma em forma de biblioteca de código.

A Antidote foi desenvolvida pela Signove, - empresa situada em Campina Grande – Paraíba, fundada em 2009 - é uma empresa de software brasileira focada em pesquisa e desenvolvimento de tecnologias para a saúde conectada. A Signove teve um importante papel no desenvolvimento do HAM, auxiliando o projeto com orientações sobre a arquitetura do sistema, além de gerar requisitos para o software.

[6] Disponível em: http://www.signove.com/plataforma-sighealth/. Acesso em outubro de 2016.

Com o uso da Antidote, o HAM passa a atender padrões de qualidade internacionais, o que traz credibilidade para ferramenta. No entanto, seu principal objetivo é prover interoperabilidade computador-dispositivo.

2.1.2 Continua Health Alliance

Assim como o HAM que segue a norma ISO/IEEE 11073, os dispositivos que quiserem se comunicar com ele também precisam estar sob essa norma de comunicação. E para dar mais confiabilidade para os dispositivos que implementam esse tipo de tecnologia, existe um selo de certificação garantido pelo grupo Continua Alliance que escolheu tanto ISO/IEEE 11073 como principal biblioteca quanto as tecnologias de transporte, como por exemplo *Bluetooth*, *Bluetooth Low Energy*, e USB. De acordo com (AntidoteProgramGuide, 2011, p. 15) a “Continua Alliance é uma associação sem fins lucrativos que promove a padronização de dispositivos pessoais de saúde, vislumbrando um padrão de mercado, acessível e facilmente conectável a sensores. ”.

2.4 OpenICE

O OpenICE é um ambiente clínico integrado (*Integrated Clinical Envorinment*) (ICE), utilizado para centralizar dados gerados por dispositivos hospitalares interligados através de uma rede local. Ele se baseia na norma ASTM F2761-2009 que define um ICE genérico, e é mantido pelo time de desenvolvedores da MD PnP, uma entidade estadunidense que se preocupa em promover inovação na área da saúde conectada, através de programas de pesquisas focados em *safety*, e cuidados clínicos.

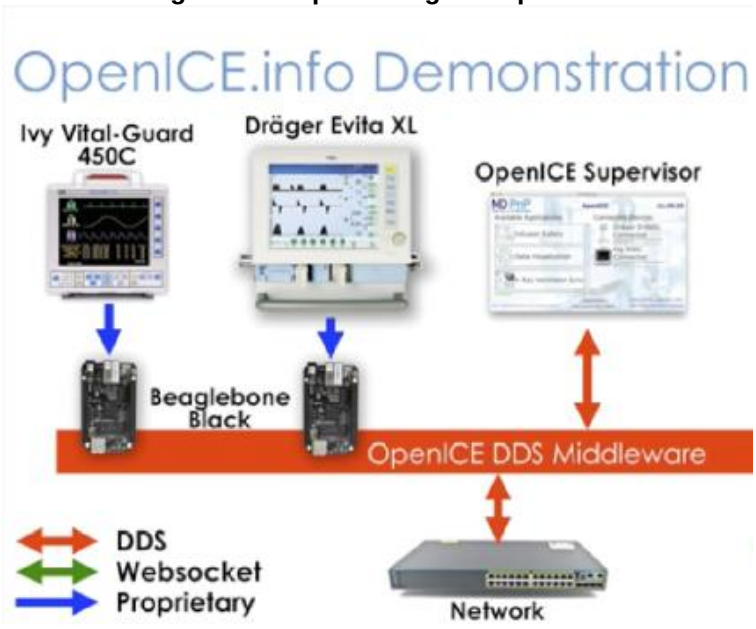
Um dos movimentos tecnológicos que vem tomando mais força atualmente é o da internet das coisas. Segundo (TOMAS, 2014), a ideia básica deste conceito é a presença de um conjunto de objetos (things) - como Radio-Frequency IDentification (RFID), sensores, atuadores - que, por meio de mecanismos de endereçamento único (como a Internet), são capazes de interagir e cooperar uns com os outros para alcançar objetivos em comum. Podemos ver no OpenICE uma exemplificação clara da internet da coisas aplicada ao universo da saúde conectada. Sendo o OpenICE uma ferramenta que gerencia/monitora dispositivos hospitalares, podemos fazer

assim um paralelo enxergando esses dispositivos como as “coisas” do nosso cenário.

Com o OpenICE temos uma ferramenta que funciona de três modos, como adaptador, simulador de dispositivos, e supervisor. Vamos explorar cada um desses modos a seguir.

Como adaptador não usamos diretamente a versão para desktop da ferramenta, mas sim uma imagem para uma *single-board* como Arduino, Rasperry ou Beaglebone. Vale ressaltar que uma imagem se trata de um sistema operacional customizado, compilado com ferramentas e recursos para atender usos específicos. No caso da imagem disponibilizada pela equipe de desenvolvimento do OpenICE, traz consigo a distribuição do Linux Debian 7.8, o kit de desenvolvimento Java versão 8u33, o OpenICE, e o protocolo NTP. Com essa imagem instalada em uma *single-board* podemos usá-la como interface de comunicação que ficará entre o dispositivo médico e a rede. O esquema está ilustrado na figura 3, onde temos uma Beaglebone fazendo o papel de interface de comunicação, traduzindo os dados vindos do dispositivo em formato proprietário, em dados que possam ser enxergados e rastreados pelo supervisor.

Figura 3 - Arquitetura geral OpenICE



Fonte: Site do OpenICE[7]

[7] Disponível em: https://vimeo.com/116917809. Acesso em outubro de 2016

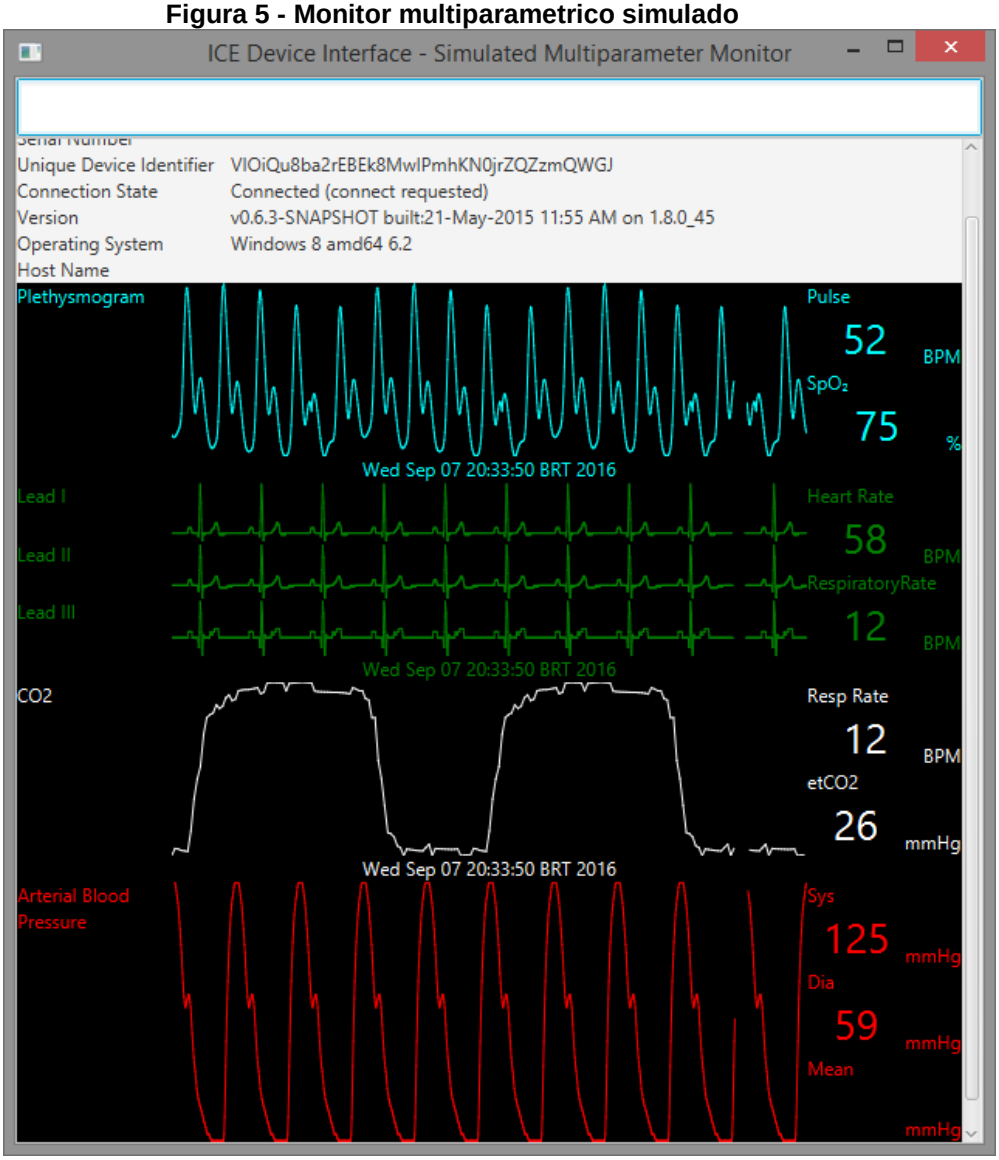
Já enquanto simulador o OpenICE virtualiza dispositivos de uma lista fornecida pela própria ferramenta, como pode ser visualizado na Figura 4:

Figura 4 - Lista de dispositivos que podem ser simulados

Fonte: Elaborada pelo autor

Depois de selecionar o tipo de dispositivo desejado é possível dar início à simulação, que pode ser vista executando na Figura 5, nesse caso temos um monitor multiparametrico simulado exibindo os dados gerados e enviando eles para a rede. Esses dados envolvem medições, como pulso, pressão sanguínea, taxa de respiração entre outros, assim como questões relacionadas a identificação do simulador, como seu ID, o sistema operacional no qual ele está executando, versão e etc. E variam de acordo com o dispositivo.

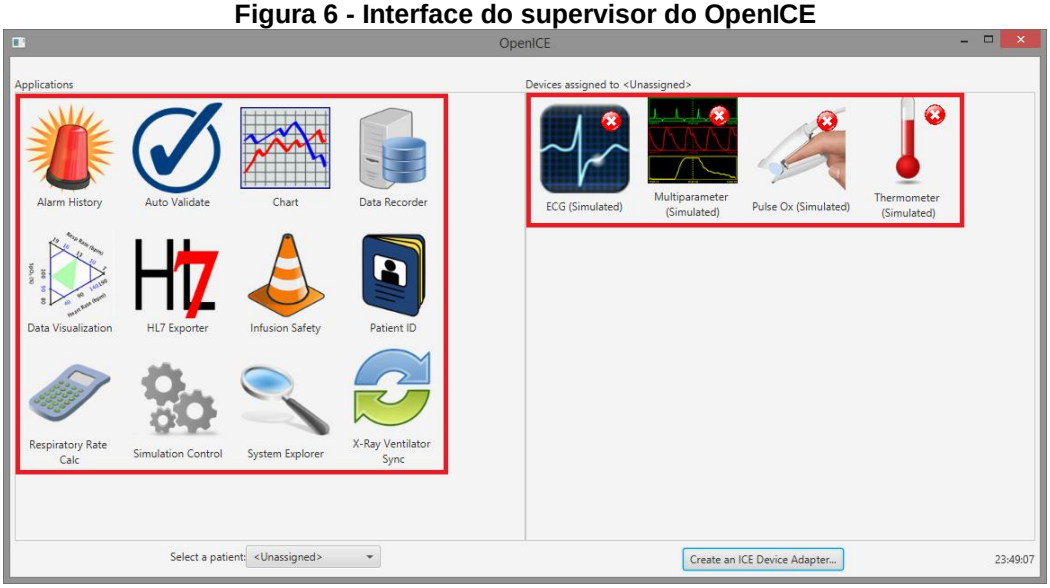
Figura 5 - Monitor multiparametrico simulado



Fonte: Elaborada pelo autor

O modo supervisor do OpenICE é onde encontramos a ferramenta na sua forma mais completa, nesse modo temos um software capaz de monitorar os dispositivos conectados à rede, sejam eles dispositivos reais ou simulados pela própria ferramenta. O supervisor traz consigo uma suíte de aplicativos que permitem realizar vários tipos de processamentos com os dados recebidos dos dispositivos. Vejamos a Figura 6 para mais detalhes, do lado esquerdo da interface do supervisor temos todas as aplicações disponíveis.

Figura 6 - Interface do supervisor do OpenICE



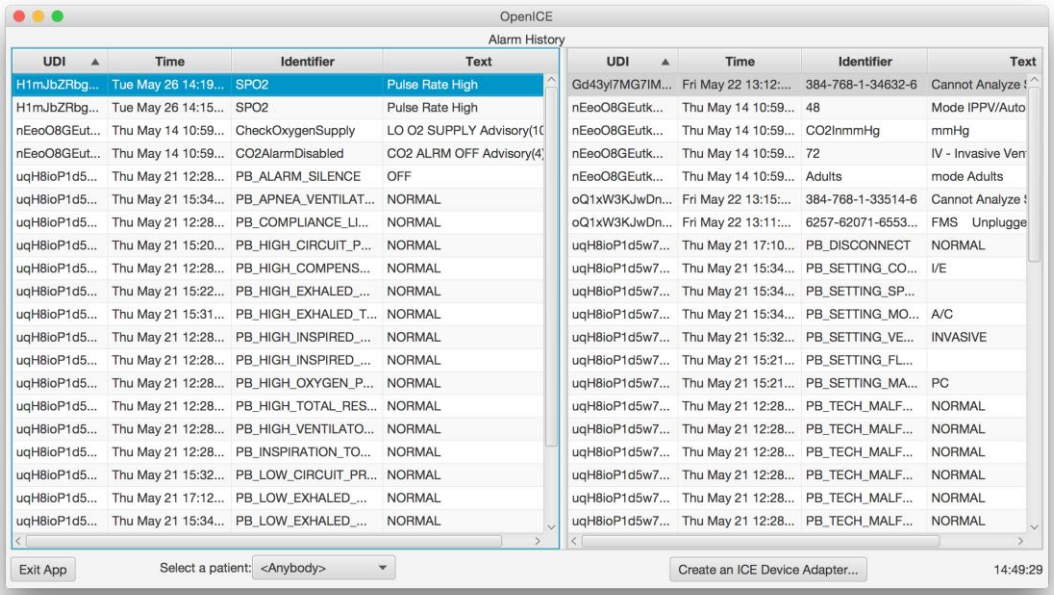
Fonte: Elaborada pelo autor

A seguir será detalhado algumas dessas aplicações, no intuito de exemplificar as capacidades da ferramenta se tratando de monitoramento, reiterando assim o porquê dela ter sido escolhida para ser o principal sistema dessa interface:

Alarm History

Essa aplicação é responsável por exibir todos os alarmes que foram disparados durante o monitoramento dos dispositivos. Podemos visualizar sua interface na Figura 7. No painel do lado esquerdo é disparado alarmes associados ao estado do paciente como, por exemplo, alguma medição fora do normal. Já o painel direito está relacionado com o dispositivo em si, quando ele não consegue realizar alguma analise, ou quando foi desconectado do paciente, são exemplos de alarmes presentes nesse painel.

Figura 7 - Janela da ferramenta Alarm History



Fonte: Site do OpenICE[8]

Auto Validate

Com essa aplicação podemos exibir em formato de histogramas dados validados, e emitidos por vários dispositivos. Nessa aplicação o usuário tem a capacidade de visualizar dados de diferentes dispositivos que têm a mesma medição fisiológica em comum, assim o usuário pode comparar e se possível validar a acurácia dos dados recebidos. Veja a Figura 8 para mais detalhes.

[8] Disponível em: https://www.openice.info/docs/3_apps.html. Acesso em outubro de 2016

Figura 8 - Janela da ferramenta Auto Validate

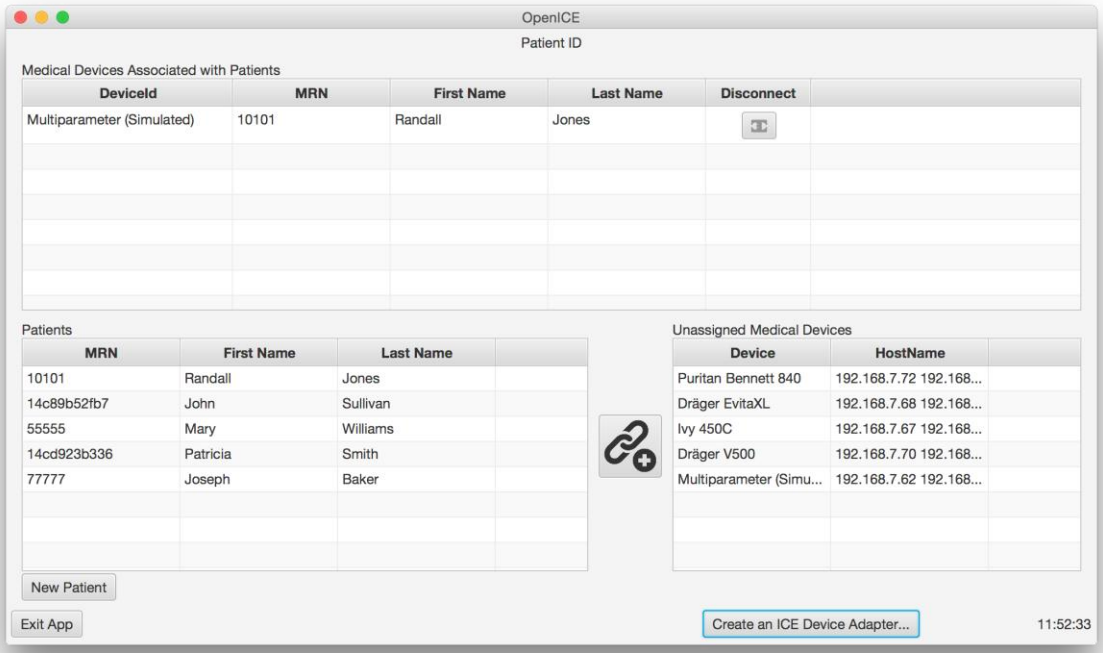
Fonte: Site do OpenICE[9]

Patient ID

Essa aplicação ajuda na associação de um ou mais dispositivos a um determinado paciente. Nesse caso, cada paciente tem seu contexto no supervisor do OpenICE, de modo que depois que um dispositivo for conectado a um paciente, o dispositivo passará a existir apenas no contexto do paciente. Observando a Figura 9, podemos ver que no painel maior acima da interface temos quais dispositivos estão conectados aos pacientes no momento. O painel ao lado inferior esquerdo exibe todos os pacientes cadastrados no supervisor. Por fim, no lado inferior direito mostra todos os dispositivos que estão “vivos” na rede, mas não foram conectados a nenhum paciente.

[9] Disponível em: https://www.openice.info/docs/3_apps.html. Acesso em outubro de 2016

Figura 9 - Janela da ferramenta Patient ID



Fonte: Site do OpenICE[10]

Voltando a falar da interface do supervisor, mostrada na Figura 6, vamos focar agora no painel à direita onde fica concentrado os dispositivos que estão online na rede, mas que não foram conectados a nenhum paciente, também conhecidos como dispositivos do contexto *Unsigned*. Ao clicarmos em qualquer dispositivo ativo nesse painel, como um monitor multiparametrico, por exemplo, uma janela semelhante à exibida na Figura 5 será mostrada, permitindo assim o monitoramento individual. Já após um dispositivo ser conectado a um paciente – através da aplicação Patient ID - ele deixa de existir no contexto *Unsigned*, e passa a existir no contexto do paciente. Isolando assim cada dispositivo por paciente. Um exemplo desse tipo de isolamento pode ser visto na Figura 10.

Algo que chama atenção com relação ao supervisor do OpenICE é a capacidade de encontrar outros dispositivos ativos na rede de forma automática, sem ser necessário configurações de IPs ou portas para que ocorra essa comunicação. Essa característica do OpenICE o torna uma tecnologia *Plug And Play,* sendo que essa e outras *features* só são possíveis graças a uma tecnologia

[10] Disponível em: https://www.openice.info/docs/3_apps.html. Acesso em outubro de 2016

muito utilizada em internet das coisas que é o middleware DDS. Detalharemos essa tecnologia melhor mais adiante.

Figura 10 - Monitor multiparametrico isolado no contexto de um paciente

Fonte: Elaborada pelo autor

2.4.1 DDS

Construir aplicações interoperáveis e distribuídas se torna algo difícil quando levamos em conta a heterogeneidade das tecnologias envolvidas, visto que existe a necessidade de lidar com diferentes sistemas operacionais, hardwares e protocolos de rede. Para minimizar esse tipo de problema, existe soluções em middleware como o DDS. O DDS (do inglês Data Distribution Service) é um middleware de alta performance voltado para comunicação de aplicações conectadas. Com o DDS criamos uma camada de abstração entre a aplicação e o sistema operacional, como pode ser visto na Figura 11, proporcionando assim isolamento entre a aplicação, e a lógica de comunicação.

Figura 11 - Camadas de uma aplicação DDS

Fonte: Elaborada pelo autor

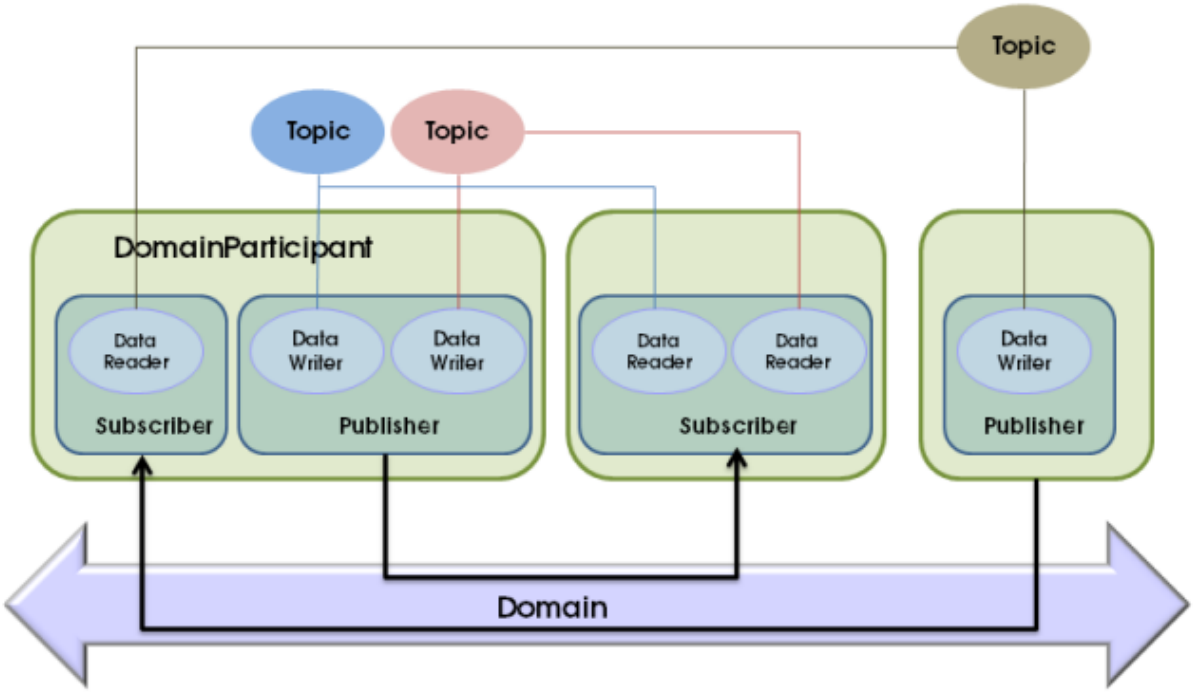
O isolamento oferecido pelo DDS enquanto middleware traz consigo uma série de benefícios como redução no tempo, e custo do desenvolvimento de uma aplicação. Já que os desenvolvedores podem se concentrar apenas na construção da aplicação evitando o desenvolvimento de módulos de conectividade. Consequentemente, essa separação de papéis proporcionada pelo middleware auxilia na manutenabilidade da aplicação, devido ao uso de código já testado, documentado e principalmente desacoplado do restante do projeto.

Um grande diferencial do DDS está no uso do conceito de ambientes centrados em dados, onde o desenvolvedor pode customizar e controlar os tipos de dados a serem enviados pela sua aplicação distribuída. Essa característica do middleware permite que ele seja mais rápido, e seguro que outras soluções apresentadas no mercado centrados em mensagens genéricas para realizar a comunicação. Outra característica que é muito apreciada no DDS é a capacidade dos nós que aplicam esse tipo de tecnologia se descobrirem automaticamente, e a partir disso passam a se comunicarem, evitando que qualquer tipo de configuração prévia por parte do desenvolvedor, ou do usuário da aplicação seja necessária. Vale também citar a flexibilidade na mudança de comportamento que o middleware prover através de *Quality of Service* (QoS). Com a QoS podemos alterar algumas políticas das entidades do DDS, modificando assim seu funcionamento para que seja possível adapta-lo melhor à sua aplicação. Alguns tipos de QoS permitem, por exemplo, mudar o tipo de comunicação para confiável, salvar dados em um histórico antes de envia-los e até mesmo filtrar os tipos de dados que serão recebidos.

2.4.2 Entidades do DDS

Por trás de todas as facilidades trazidas pelo DDS há uma grande infraestrutura composta pelo o que a literatura chama de entidades. Cada entidade desempenha um papel especifico, e conta com suas próprias QoS no DDS. Dedicaremos os próximos tópicos a listar quais são essas entidades e quais são seus papeis na arquitetura. Para visualizar melhor essas entidades na arquitetura do DDS, levaremos em consideração a Figura 12.

Figura 12 - Entidades do DDS

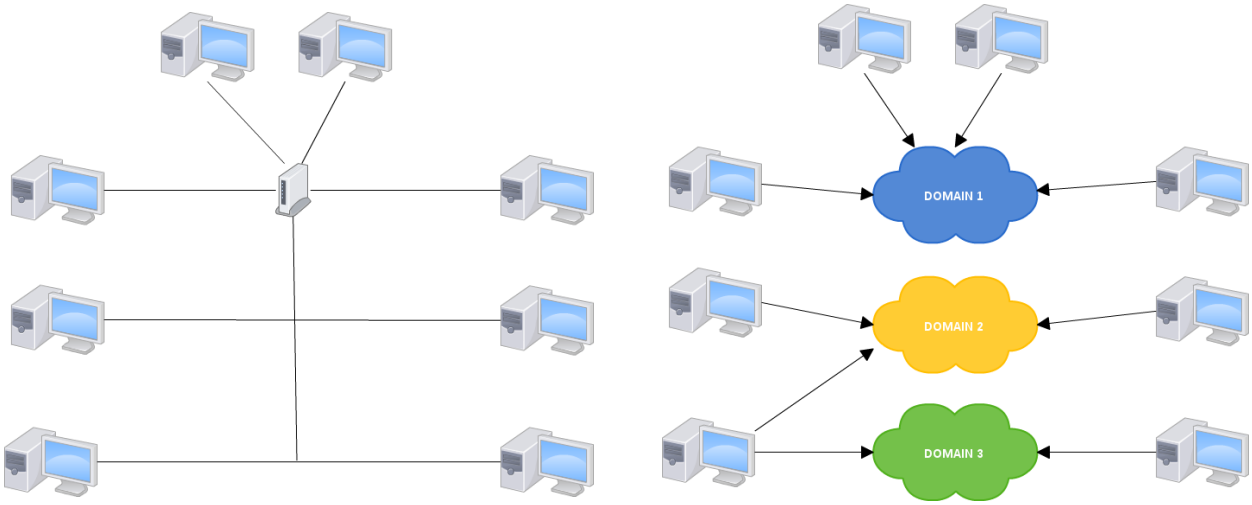


Fonte: Manual do Connext DDS[11]

Domain: Domain é uma espécie de rede virtual isolada por onde os dados vão trafegar, a Figura 13 ilustra o papel o seu papel. Ao lado esquerdo temos a representação de uma rede padrão de computadores onde todos os computadores participam da mesma rede, e já ao lado direito temos a mesma rede separada logicamente por *Domains* do DDS. Esse isolamento garante que um dado enviado em um *Domain* não possa ser encontrado em qualquer outro trazendo segurança aos dados.

[11] Disponível em: https://community.rti.com/static/documentation/connext-dds/5.2.3/doc/manuals/connext_dds/RTI_ConnextDDS_CoreLibraries_UsersManual.pdf. Acesso em outubro 2016

Figura 13 - a) Uma rede de computadores padrão. b) A mesma rede separada por Domains



Fonte: Elaborada pelo autor

DomainParticipant: Essa entidade representa a instância da aplicação no *Domain*, e a partir dela outras entidades fundamentais como *Publishers* e *Subscribers* são instanciadas.

Topic: O *Topic* está associado ao tipo de dado que vai ser transmitido no *Domain*. Observando a Figura 12, podemos notar que os *Topics* interconecta aplicações de forma que uma aplicação só pode se comunicar com outra se os seus *DataWriters* e *DataReader* trabalharem com o mesmo tipo de dado, ou melhor, com o mesmo *Topic*.

Publisher: Encapsula as funcionalidades referentes ao envio de dados. Serve também de involucro para agrupar um ou mais *DataWriters*, entidades pertencentes ao Publisher responsável pelo o envio final dos dados. Sendo que alterações nas configurações de um Publisher reflete em seus *DataWriters*.

Subscriber: Encapsula as funcionalidades referentes ao recebimento de dados. Serve também de involucro para agrupar um ou mais *DataReaders*, que são entidades pertencentes ao *Subscriber* responsável pelo o recebimento final dos dados. Sendo que alterações nas configurações de um *Subscriber* reflete em seus *DataReaders*.

DataWriter: Uma aplicação usa DataWriters para enviar dados. Além disso, você pode ter mais de um DataWriter para um determinado *Topic* em uma única aplicação.

DataReader: Uma aplicação usa *DataReader* para receber dados. Além disso, você pode ter mais de um *DataReader* para um determinado tópico em uma única aplicação.

2.4.3 Connext DDS

Dentre as várias distribuições do DDS a escolhida foi a Connext DDS, distribuição essa mantida pela empresa Real-Time Inovations (RTI) que inclusive foi uma das idealizadoras do middleware junto a Thales Group. O Connext DDS, por ser uma das primeiras plataformas desse gênero possui uma grande comunidade de usuários o que facilita o suporte à tecnologia, outra diferença do Connext DDS é a suite de aplicações que ele carrega consigo, como por exemplo, a ferramenta *Code Generator* detalhada a seguir.

2.4.3.1 A ferramenta Code Generator e a IDL

O *Code generator* talvez seja a ferramenta mais importante da suíte de ferramentas do Connext DDS, visto que ela permite gerar a infraestrutura de comunicação do DDS (entidades) em diferentes linguagens de programação, e arquiteturas. Para realizar essa tarefa o *Code Generator* recebe como entrada um arquivo do tipo *Interactive Data Language* (IDL), semelhante ao presente no apêndice A. Esse arquivo define os *Topics* (Estruturas de dados) que serão enviados no *Domain* pelos *Publishers*. Além disso, também gera parte da infraestrutura DDS presente no tópico 2.5.1 desse trabalho.

2.6 D-Bus

Uma ferramenta que se mostrou necessária na construção da interface de comunicação foi o D-Bus. O D-Bus é um mecanismo de comunicação inter-processos que permite que dois programas diferentes, (i.e processos) se comuniquem. Ao estabelecermos uma conexão D-Bus entre dois processos eles passam a se comunicarem por uma interface genérica que abstrai questões intrínsecas a linguagens como tipos e estruturas de dados, fazendo com que a troca de mensagens entre dois programas seja possível.

2.7 Engenharia reversa em softwares

Se baseando em (SAPAGE et al., 2005) e (STEFANELLI, 2009) podemos definir engenharia de software como uma técnica utilizada em vários campos do conhecimento na tentativa entender o funcionamento de um produto finalizado ou não. Em software essa técnica, busca extrair detalhes da implementação com objetivos como:

- Primordialmente, entender um software existente para replicá-lo.
- Encontrar falhas de segurança. Uma pessoa com um diferente background analisando um código escrito pode encontrar falhas que passaram pelo desenvolvedor e testadores.
- Documentar um software legado, ou que simplesmente não tenha documentação disponível.

Contudo, não podemos aplicar engenharia reversa em todos os casos. Muitos softwares são protegidos legalmente, e a engenharia reversa fere os direitos dos seus proprietários, por isso há a necessidade de usar essa técnica de modo consciente.

Em softwares livres, podemos ver a engenharia reversa como uma boa forma de aprender mais sobre uma determinada linguagem ou conjunto de tecnologias. Analisar código de programadores mais experientes pode ser uma experiência enriquecedora abrindo os horizontes para desenvolvedores iniciantes.

3. ENTENDENDO A INTERFACE DE COMUNICAÇÃO

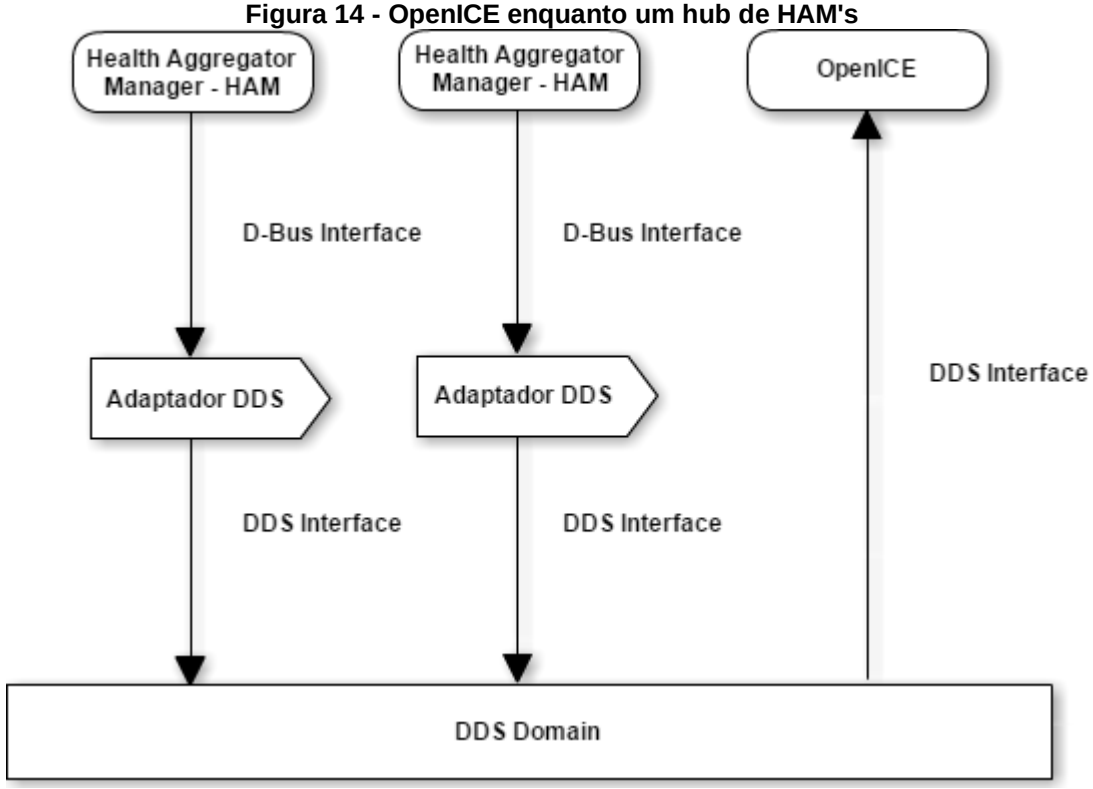
Esse capítulo terá como finalidade descrever como foi construída a interface de comunicação. O produto final dessa interface foi um software que funciona como um adaptador DDS, dando a capacidade de agregadores de dados em saúde como o HAM de se comunicar com o OpenICE. Durante este capítulo visualizaremos onde foi empregada cada tecnologia, sistema, e técnica citada do capítulo anterior na construção do adaptador.

3.1 A interface de comunicação

Os dois sistemas alvos desse trabalho de integração desempenham uma função à primeira vista semelhantes, que é o monitoramento de dados provenientes de dispositivos médicos. Contudo, para que essa função seja exercida em sua totalidade, ambos os sistemas precisam garantir interoperabilidade com outros dispositivos. No HAM, a interoperabilidade é atingida através da biblioteca Antidote, que é uma implementação da norma ISO/IEEE 11073. Já no caso do OpenICE, a responsabilidade cai sob o *adaptor DDS*, implantado através de uma *singleboard*. Sendo o objetivo desse trabalho especificar uma interface de comunicação entre esses dois sistemas, lidamos aqui com um outro tipo de interoperabilidade. A interoperabilidade entre sistemas. De um lado temos o HAM, um sistema desenvolvido em QT creator para a arquitetura Linux, já do outro o OpenICE um sistema feito em Java, que se vale da JVM para executar em diversas plataformas.

Com esse panorama em mente, esse trabalho propõe especificar uma interface com base no DDS, visto que o nosso ICE foi construído em cima dessa tecnologia. Assim como o supervisor do OpenICE encontra dispositivos “vivos” na rede, buscamos definir como o supervisor pode encontrar e monitorar outro sistema como o HAM na rede. Para fazer isso, usamos a mesma ideia do *adaptor* DDS para dispositivos médicos mostrado na Figura 3, só que dessa vez aplicado em um software que é o HAM. Esse esquema pode ser visualizado na Figura 14. Onde cada HAM terá seu adaptador DDS

Figura 14 - OpenICE enquanto um hub de HAM's



Fonte:Elaborada pelo autor

3.2 A construção do adaptador DDS

A viabilidade da construção do adaptador[12] DDS surgiu após termos conhecimento de um software disponibilizado pela equipe de desenvolvimento do OpenICE. Ele se chama Hello OpenICE, é uma versão enxuta do supervisor, e simulador de dispositivos do OpenICE. Ele foi escrito para funcionar através de linha de comando abstraindo toda a complexidade trazida por uma interface gráfica. Aplicando engenharia reversa nesse software para entender seu funcionamento, se viu uma abertura para transformá-lo em um adaptador DDS para o HAM. Assim, usando o Hello OpenICE como base, o adaptador DDS foi desenvolvido, e esse desenvolvimento se deu em três etapas:

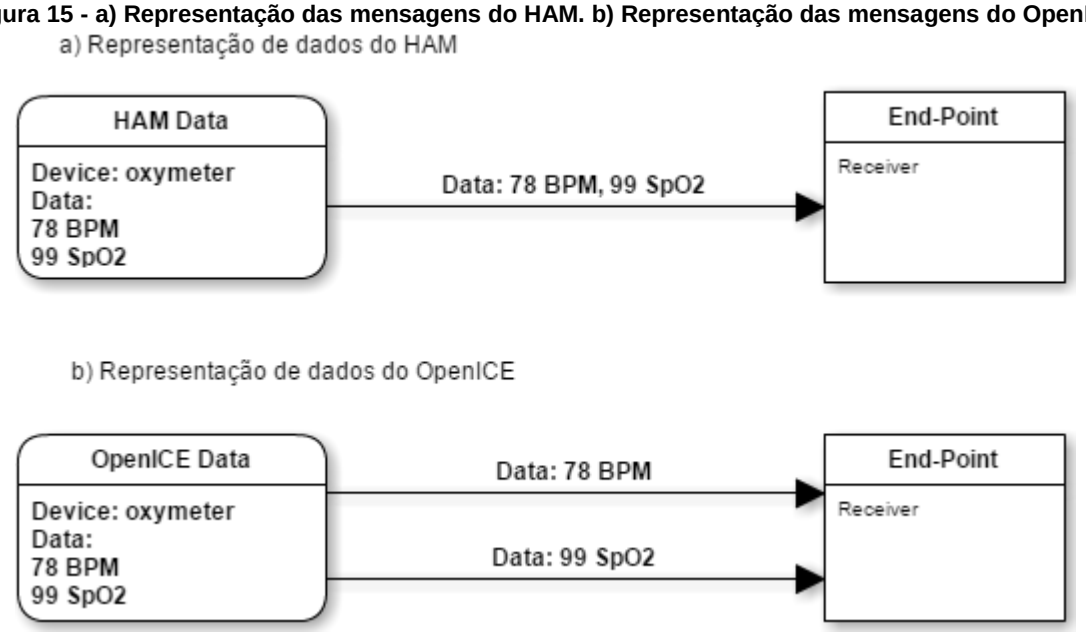
i) A definição de uma IDL própria para o HAM

O HAM e o OpenICE tratam os dados com a mesma terminologia, provida pela norma ISO/IEEE 11073. Isso significa dizer que da mesma forma que o HAM enxerga o valor, a unidade de uma medição do nosso ICE também enxerga.

[12] https://github.com/alissonssz/DDS-Adaptor-

Contudo, isso não é o suficiente para que as partes se entendam, pois o HAM é completamente imerso na norma ISO/IEEE 11073. Isso inclui até a forma que se representa os dados empacotados, já o OpenICE empacota os dados através de uma estrutura própria definida pelos *Topics* da aplicação, onde cada medição é uma mensagem diferente. Veja a Figura 15 para mais detalhes. Precisamos então que um dos lados se adapte ao outro, e depois de análise, foi escolhido o modelo de dados do HAM, devido ao fato que não precisaríamos tratar a quantidade de mensagens que devem ser enviadas dependendo de cada dispositivo, como mostrado na Figura 15b.

Figura 15 - a) Representação das mensagens do HAM. b) Representação das mensagens do OpenICE



Fonte: Elaborada pelo autor

Depois que houve a definição de modelo de dados, o próximo passo a seguir foi definir um novo *Topic* para o adaptador DDS. Lembrando que o *Topic* é a estrutura de dados que será trocada entre os nós participantes do *Domain*. Para a definição dessa estrutura, a IDL original do OpenICE foi reutilizada, porém com a adição de um novo *Topic*, o HAM_Device. O trecho de código contendo esse incremento na IDL pode ser visto na Figura 16.

Figura 16 - *Topic* específico para o HAM

```
302   struct HAM_Device {
303    SystemID unique_system_id; //@key
304    VendorMetricIdentifier vendor_metric_id; //@key
305    Model device_model;
306    Specializations specialization_code;
307    UnitIdentifierList unit_id; //@key
308    UnitSymbolList unit_symbol;
309    ValueMeasurementList values;
310    DateList measurement_date;
311    TimeList measurement_time;
312  }; //@top-level true //@Extensibility MUTABLE_EXTENSIBILITY
```

Fonte: Elaborada pelo autor

Esse topic é composto pelos seguintes campos:

unique_system_id: Um id que representa unicamente o dispositivo

vendor_metric_id: Nome do fabricante do dispositivo

device_model: Modelo do dispositivo

unit_id: Lista com os códigos de unidades de cada medição

unit_symbol: Lista de símbolos de cada unidade, por exemplo: bpm, spo2, ml

values: Lista com os valores de cada medição

measurement_date: Data da medição

measurement_time: Hora da medição

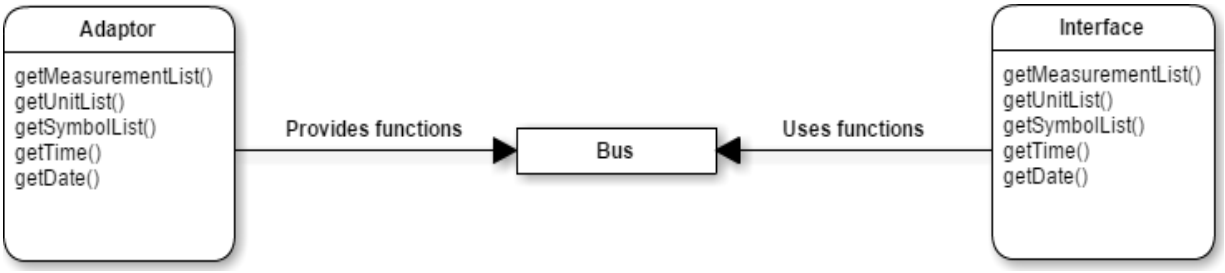
ii) Desenvolvimento de uma interface de comunicação D-Bus entre o Adaptor DDS e o HAM.

Por padrão, o adaptador DDS funciona como um processo paralelo ao HAM, sendo que o HAM tem a necessidade de utilizar algumas funcionalidades do adaptador com a finalidade de se comunicar com o OpenICE. Para lidar com essa questão, foi utilizado um mecanismo de comunicação entre processos, o D-Bus.

Para essa interface de comunicação precisamos fazer algumas alterações em ambos os softwares, que consistem em implantar um módulo de comunicação D-Bus em cada software. Esses módulos diferem por meio de suas funções. No HAM o módulo oferece os recursos através de um ***adaptor*** (vamos deixar essa palavra em inglês para evitar confusões com o adaptador DDS). O *adaptor* publica alguns dos métodos do HAM em um *Bus* para que o adaptador DDS possa utilizá-los. Do outro lado, temos o adaptador DDS com uma ***interface*** que espelha os métodos

publicados pelo HAM. Com esses componentes implementados somos capazes de fazer com que os dois softwares se comuniquem. A Figura 17 ilustra esse esquema.

Figura 17 - Funcionamento do D-Bus no Adaptador DDS

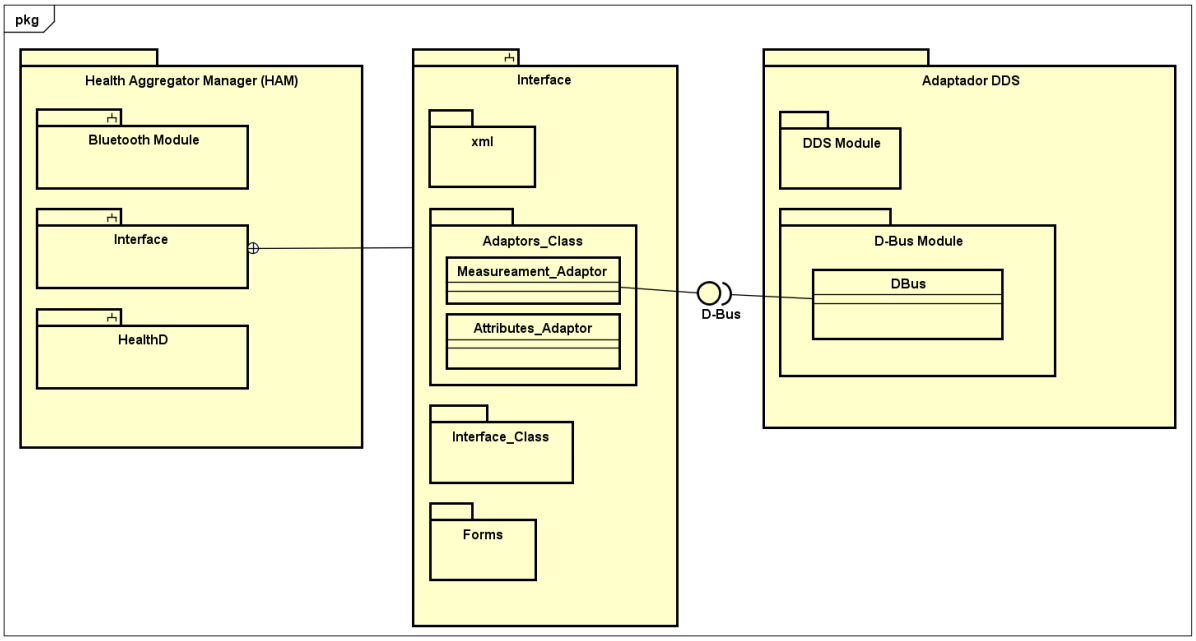


Fonte: Elaborada pelo autor

Quando o adaptador DDS quer, por exemplo, pegar as medições disponíveis no HAM atualmente, ele chama o método getMeasurementList(). Esse método é então buscado no *Bus*, se ele estiver presente, será chamado no programa que o disponibilizou no *Bus*. O retorno é devolvido então para o “método espelho“ no adaptador DDS. E assim finalizamos o ciclo de comunicação D-Bus.

A estrutura final dos elementos participantes da interface D-Bus pode ser encontrada na Figura 18. Nessa figura temos definidos os dois elementos que irão interagir nessa interface o HAM e o Adaptador DDS. Como pode ser visto, HAM é composto por três subsistemas o Bluetooth Module é responsável por toda lógica de comunicação via bluetooth do sistema, a Interface é outro subsistema que guarda classes relacionadas a lógica de negócio da aplicação, além de classes relacionadas a interface gráfica do sistema, e por fim o HealthD encarregado de gerenciar junto ao Bluetooth Module a comunicação do HAM em conformidade com a norma ISO/IEEE 11073. Ao olharmos dentro do subsistema Interface encontraremos alguns pacotes que o compõe, são eles: xml, Adaptors_Class, Interface_Class e Forms. O pacote Adaptors_Class merece uma atenção especial, pois nele que está presente o adaptor D-Bus apresentado na Figura 17. Já o Adaptador DDS é constituído de dois pacotes, o primeiro é o DDS module, responsável por toda a lógica de comunicação do DDS, já o outro D-Bus module possui uma classe capaz de agir como a Interface da Figura 17.

Figura 18 - Diagrama estrutural da comunicação D-Bus da interface



Fonte: Elaborada pelo autor

iii) Encontrando o HAM com o OpenICE

Como já dito anteriormente, uma das grandes vantagens do OpenICE está na sua capacidade de encontrar dispositivos na rede por contra própria. E essa capacidade é passível de ser implementada no nosso adaptador DDS, fazendo com que o HAM seja encontrado pelo OpenICE, como qualquer outro dispositivo conectado à rede DDS. Para entender como isso foi possível, vamos começar pelo caso base. O OpenICE encontra dispositivos conectados ao *Domain* através de dois *Publishers* destinados à parte de descoberta. Esses *Publishers* utilizam os seguintes *Topics*:

a) DeviceIdentity

O papel do *Publisher* que utiliza esse *Topic* é anunciar a aplicação na rede, enviando informações sobre especificações do dispositivo, como modelo, fabricante, e até mesmo o ícone que vai representar o dispositivo na interface do OpenICE. A Figura 19 mostra em código esse *Topic*.

Figura 19 - Topic DeviceIdentity em código

```
public String unique_device_identifier = ""; /* maximum length = (64) */
public String manufacturer = ""; /* maximum length = (128) */
public String model = ""; /* maximum length = (128) */
public String serial_number = ""; /* maximum length = (128) */
public ice.Image icon = (ice.Image) ice.Image.create();
public String build = ""; /* maximum length = (128) */
public String operating_system = ""; /* maximum length = (128) */
```

Fonte: Elaborada pelo autor

b) DeviceConnectivity

A principal função dessa entidade é guardar o estado do dispositivo ao ser encontrado pelo OpenICE, que varia entre desconectado, conectando, negociando, conectado e desconectando. A Figura 20 mostra em código esse *Topic*.

Figura 20 - Topic DeviceConnectivity em código

```
public String unique_device_identifier = ""; /* maximum length = (64) */
public ice.ConnectionState state = (ice.ConnectionState) ice.ConnectionState.create();
public ice.ConnectionType type = (ice.ConnectionType) ice.ConnectionType.create();
public String info = ""; /* maximum length = (128) */
public ice.ValidTargets valid_targets = (ice.ValidTargets) ice.ValidTargets.create();
```

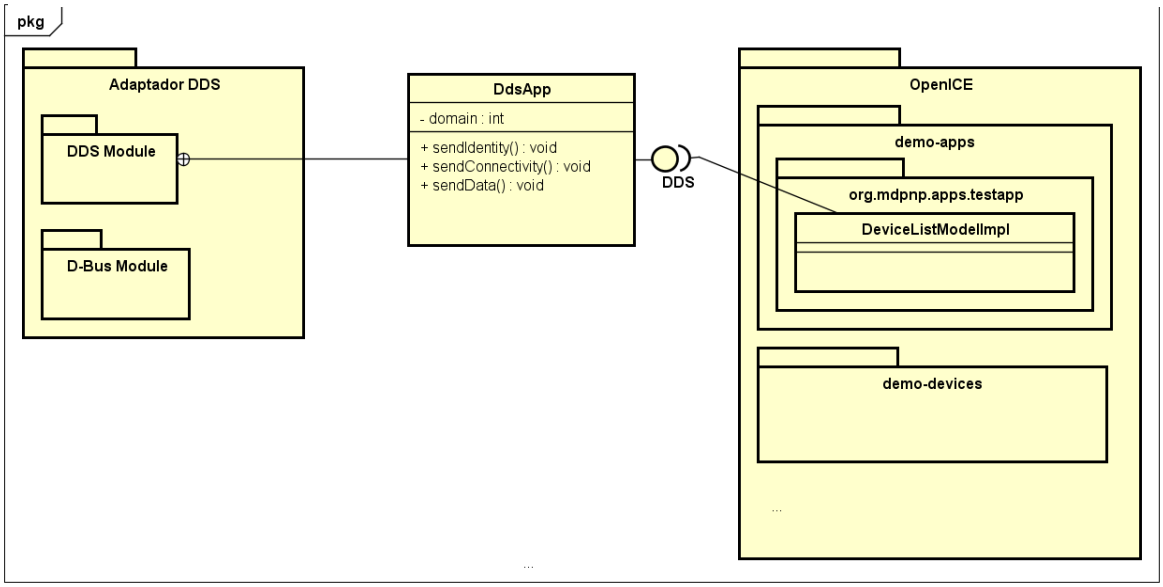
Fonte: Elaborada pelo autor

Ao observamos os *Topics* a) e b), podemos ver um campo em comum: o unique_device_identifier. É através dele que o OpenICE consegue associar mensagens fragmentadas dos *Topics* DeviceIdentity, e DeviceConnectivity a um dispositivo.

No OpenICE existem *Subscribers* configurados com os a) e b). Então, ao implementarmos *Publishers* ao nosso adaptador DDS que publicam para esses *Subscribers*, fazemos com que o HAM seja encontrado pelo OpenICE.

A Figura 21 ilustra melhor essas relações de descoberta do ponto de vista estrutural. Existe um pacote chamado DDS module composto pela classe DdsApp com *Publishers* que são responsáveis por anunciarem o Adaptador DDS na rede através dos *Topics* DeviceIdentity, e DeviceConnectivity. Com o Adaptador DDS sendo anunciado por tais *Publishers*, o OpenICE será capaz de encontrar o HAM na rede e passar a rastreá-lo.

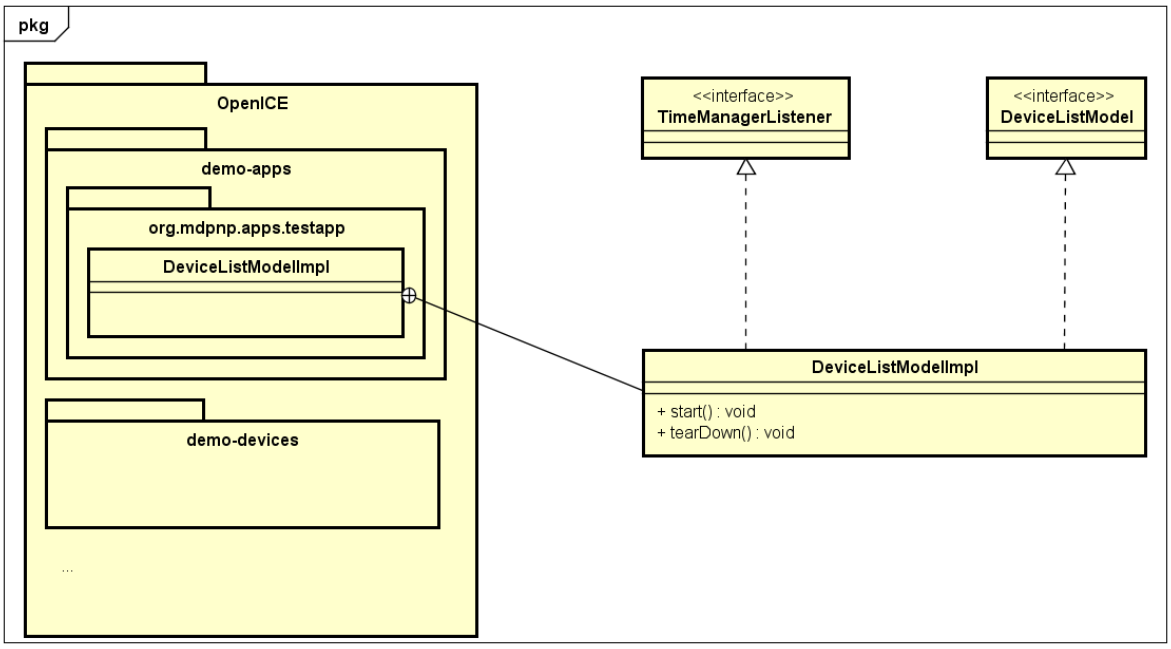
Figura 21 - Diagrama estrutural da fase de descoberta



Fonte: Elaborada pelo autor

Para entender melhor como processo de descoberta ocorre do lado do OpenICE, temos a Figura 22. No OpenICE, ao iniciar ele no seu modo supervisor, por padrão, a classe DeviceListModelImpl será instanciada e iniciará os *Subcribers* responsáveis por encontrar os dispositivos sob os *Topics* DeviceIdentity, e DeviceConnectivity. E no nosso caso, encontramos o HAM na rede.

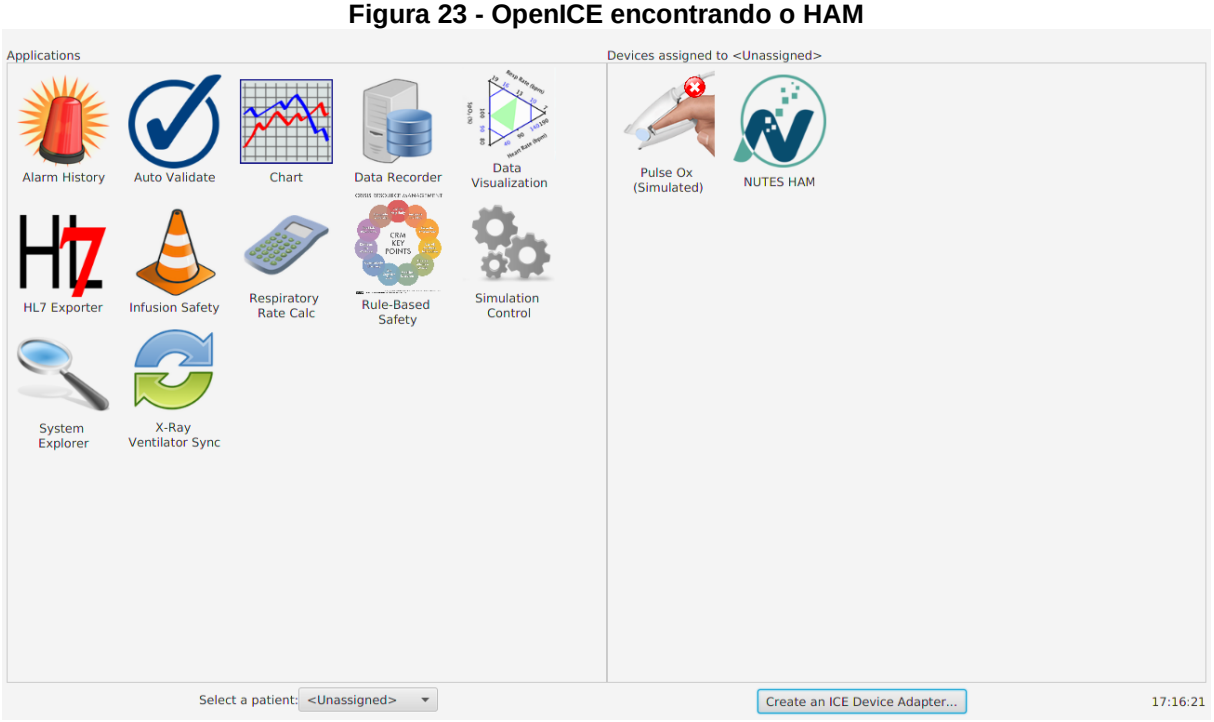
Figura 22 - Autodescoberta do OpenICE



Fonte: Elaborada pelo autor

O resultado final pode ser encontrado na Figura 23. O OpenICE encontrando o HAM, e um simulador de oxímetro. Para que tal resultado fosse alcançado foi necessário entender o sistema, e para que isso fosse possível a engenharia reversa foi utilizada. Como produto de tal técnica temos o diagrama da Figura 22.

Figura 23 - OpenICE encontrando o HAM

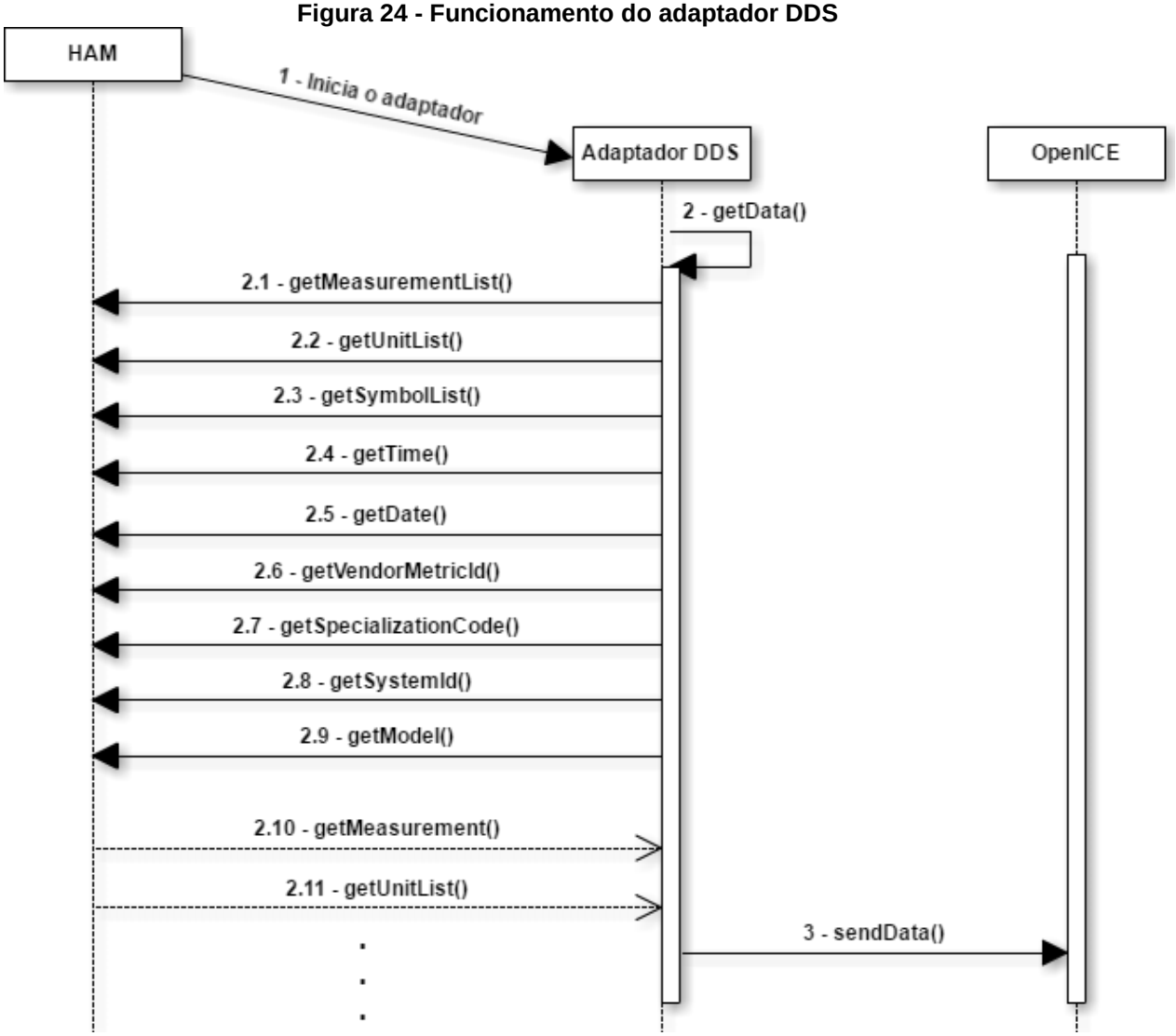


Fonte: Elaborada pelo autor

3.3 Estabelecendo a comunicação HAM-Adaptador

Vamos nos preocupar nesse momento em estabelecer uma visão macro da troca de mensagens que ocorre entre o HAM e o OpenICE com o objetivo de especificar a troca de mensagens, e mostrar o passo a passo de quando uma nova medição é recebida pelo HAM. Utilizamos a Figura 24 como guia, e acompanharemos as três etapas desse modelo.

Figura 24 - Funcionamento do adaptador DDS



Fonte: Elaborada pelo autor

Passo 1: Iniciar o adaptador

Nesse primeiro incremento, logo após receber a medição o HAM inicia o adaptador.

Passo 2: Método getData()

Através desse método, o adaptador recupera todas as informações da medição, incluindo atributos do dispositivo.

Passo 3: Método sendData()

Depois do adaptador ter todas as informações necessárias da medição, resta enviá-la para o OpenICE via DDS. O processo de envio é realizado por um *Publisher* especifico, criado exclusivamente para enviar dados com o *Topic* do HAM. Reveja a Figura 16 para mais detalhes sobre o *Topic*.

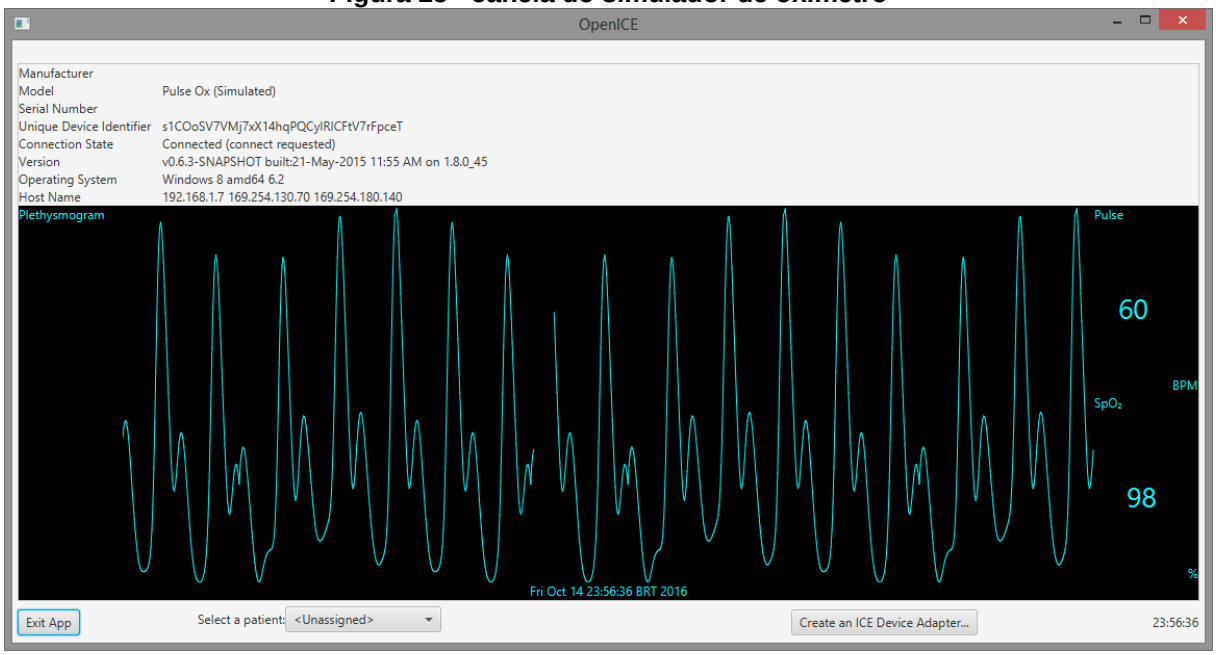
3. AVALIANDO A SOLUÇÃO

Nesse capítulo apresentaremos alguns resultados referentes a exibição de dados e quais foram as dificuldades encontradas na hora exibir dados através dessa interface de comunicação. Além disso, houve a execução de testes para validar a integração entre os sistemas. Como ocorreu os testes e quais tecnologias foram utilizadas para tais testes também estão explicitadas nesse capítulo.

4.1 A exibição dos dados.

Como mostrada na Figura 23, logo após que um dispositivo é descoberto pelo OpenICE, um ícone no painel direito representando aquele dispositivo surge. Ao clicarmos no ícone uma nova janela toma conta de toda a interface para que o usuário tenha uma prévia dos dados que estão sendo monitorados, além de exibir algumas informações do dispositivo monitorado na parte superior da janela relacionadas ao Topics DeviceIdentity e DeviceConnectivity. A Figura 25 ilustra o caso de clicarmos no ícone do oxímetro da Figura 23.

Figura 25 - Janela do simulador de oxímetro

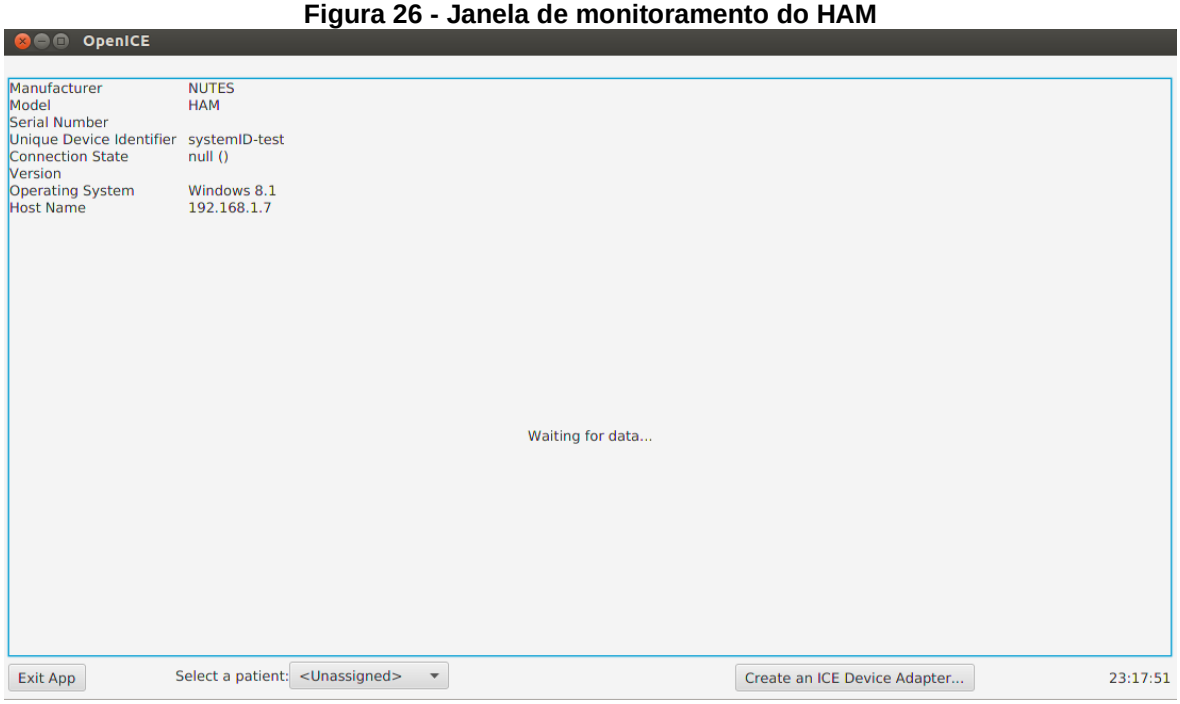


Fonte: Elaborada pelo autor

Em linhas mais técnicas, a janela da Figura 25 é definida por uma classe chamada CompositePanel. Ela é a responsável pela exibição e atualização dos

dados. Contudo, é aqui que encontramos o que talvez seja a maior barreira desse trabalho. Isso acontece porque a janela inferior onde as medições são exibidas, tem um acoplamento fortíssimo aos *Topics* padrões do OpenICE. Isso significa dizer que adicionar um novo dispositivo requer um esforço expressivo, pois exige a criação de um novo tipo de interface voltada para aquele dispositivo. Sabendo dessa questão, podemos considerar em estudos futuros a criação de um painel de interface gráfica "coringa" para resolver esse problema, o que ajudaria bastante para realização de testes, por exemplo. Essa decisão de design do OpenICE afetou a integração do HAM e o resultado disso pode ser visto na Figura 26.

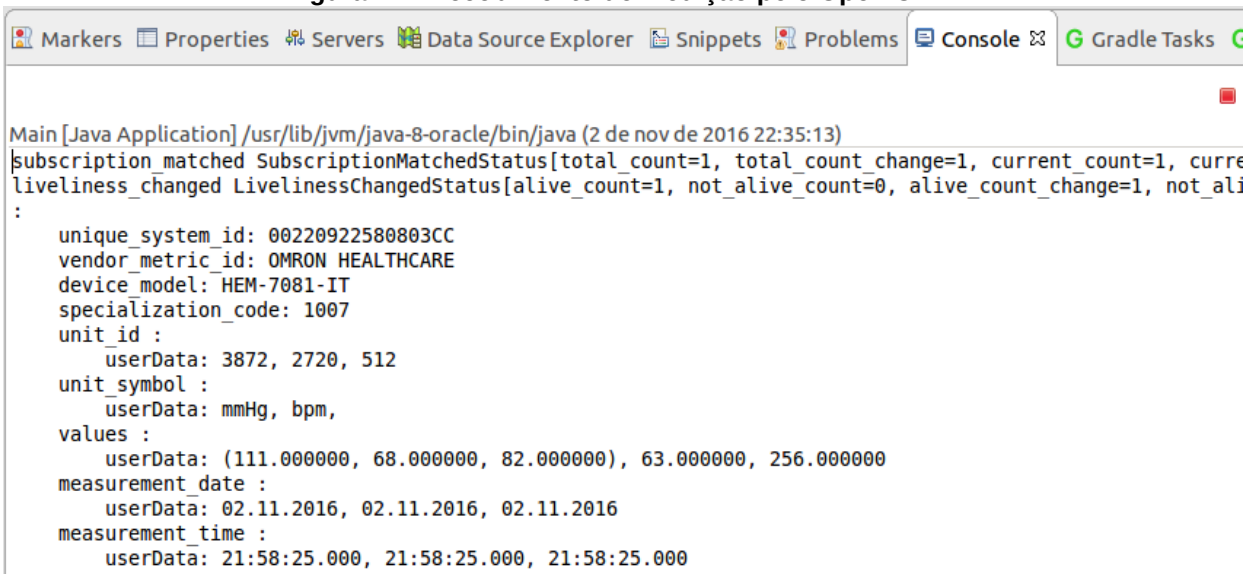
Figura 26 - Janela de monitoramento do HAM



Fonte: Elaborada pelo autor

Como podemos observar, o OpenICE consegue encontrar o dispositivo, porém no momento não há como podermos exibir em interface gráfica os dados relacionados às medições. Devido a isso, foi preciso recorrer a outros meios de validar a chegada de dados, e isso se deu inicialmente através do terminal. No nosso caso, uma das maneiras foi utilizar o terminal do ambiente de desenvolvimento, o Eclipse. A Figura 27 mostra a chegada no OpenICE de uma medição capturada pelo HAM, e que teve sua origem um dispositivo aferidor de pressão arterial. Demonstrando assim o recebimento de dados pelo OpenICE.

Figura 27 - Recebimento de medição pelo OpenICE



Fonte: Elaborada pelo autor

A Figura 27 também representa o modelo de dados que o OpenICE recebe, e que está diretamente relacionado com o *Topic* definido na Figura 16. A seguir temos uma breve descrição de cada campo:

unique_system_id: Identifica unicamente cada dispositivo.
vendor_metric_id: Fabricante do dispositivo.
device_model: Modelo do dispositivo.
specialization_code: Identifica de qual tipo de dispositivo. Ex: oximetro, balança, etc.
unit_id: Uma lista com os códigos de unidade de cada parâmetro da medição.
unit_symbol: Uma lista com os símbolos das medições.
values: Lista de valores da medições.
measurement_date: Lista com as datas de cada medição.
measurement_time: Lista com a hora de cada medição.

4.2 Validando através de testes

Um dos meios encontrados para também validar a solução apresentada nesse trabalho foi a realização de alguns testes de aceitação nos sistemas a serem integrados. Buscamos com esses testes averiguar o fluxo de dados que parte do HAM e finaliza ao chegar no OpenICE. Para o HAM, os testes realizados foram de unidade, e consistiram em verificar se o sistema realmente estava disponibilizando os dados para o nosso adaptador DDS. Já com o OpenICE, nos preocupamos em confirmar se os dados vindos do adaptador realmente estavam sendo capturados pelo OpenICE.

As tecnologias empregadas nos testes variaram devido às diferentes linguagens de programação empregadas na construção dos sistemas. Com o HAM, por ser um sistema construído em QtCreator foi necessário buscar um framework capaz de realizar testes com a linguagem. Depois de algumas pesquisas, e fracassos, o framework que melhor se adequou a necessidade foi o **gtest** da Google. No caso do OpenICE, por ser um sistema em Java, o clássico **JUnit** conseguiu cumprir bem a tarefa.

Os testes foram realizados com três dispositivos, o aferidor de pressão Onrom HEM-7081-ITZ, a balança Onrom HBF-206IT, e um terceiro dispositivo simulado disponibilizado pela biblioteca antidote que foi o oxímetro. Ao término dos testes - que foram realizados com sucesso - pudemos comprovar a eficácia da nossa interface de comunicação em levar um dado do HAM para o OpenICE. Os testes construídos encontram-se disponíveis no Apêndice B, foram completamente satisfatórios na versão concluída para este trabalho.

5. CONCLUSÃO

Procuramos especificar nesse trabalho uma interface de comunicação entre dois sistemas de monitoramento de dispositivos médicos. O Health Aggregator Manager (HAM), desenvolvido no NUTES da UEPB, e o OpenICE, que é uma implementação aberta de um Integrated Clinical Environment (ICE), mantida pela empresa estadunidense MD PnP. Ambos os sistemas são consumidores de dados, contudo o primeiro é voltado para dados de dispositivos médicos pessoais, enquanto o segundo é direcionado para dispositivos hospitalares. Visto a semelhança entre os papeis dos dois sistemas se viu a oportunidade de integra-los.

Para que a integração fosse possível técnicas de engenharia reversa de software foi aplicada ao OpenICE, isso ocorreu devido a inexistência de documentação para o mesmo. Depois de entender parte de seu funcionamento, o conhecimento acumulado foi aplicado na criação de uma interface de comunicação. A integração entre os sistemas foi alcançada através de um componente que chamamos de **adaptador DDS**. Com ele demos a capacidade do HAM se comunicar via DDS, tornando-o uma aplicação ainda mais conectada.

Apesar da interface de comunicação ser funcional, a sua total assimilação pelo OpenICE ainda não chegou ao seu ápice pois o sistema não foi construído pensando na inserção de dispositivos/aplicações com Topics diferentes, evidenciando assim seu forte acoplamento aos Topics padrões da aplicação. O resultado desse forte acoplamento foi a não exibição das medições na interface gráfica de usuário do OpenICE, restando assim validar a chegada desses dados de forma mais simplista, usando o terminal do ambiente de desenvolvimento, o Eclipse.

Esse trabalho busca contribuir de diferentes formas, principalmente no que se entende de documentação do OpenICE. Outro foco desse trabalho é explorar uma nova aplicação para o DDS, a interoperabilidade entre sistemas standalone, demonstrando assim o poder desse middleware se tratando de conectividade. E por fim, vale ressaltar o objetivo dessa integração, que foi aumentar o poder de monitoramento do OpenICE, fazendo com que ele também consiga monitorar dispositivos médicos pessoais, e consequentemente transformando o OpenICE em um hub para diferentes HAM’s que possam existir em um ambiente clínico conectado.

Para o pesquisador envolvido nesse trabalho toda a pesquisa envolvida trouxe grandes desafios, visto que a tecnologia ICE ainda é pouco explorada no meio acadêmico, exigindo assim um cuidado redobrado na hora de escrever sob tal assunto. Foi também dado ao pesquisador oportunidade de trabalhar com sistemas complexos como o OpenICE, que foi desenvolvido por profissionais experientes mostrando-o assim recursos avançados da linguagem de programação Java. E ter contato com o código fonte de tal sistema enriqueceu seus conhecimentos sobre programação, principalmente no que se entende sobre aplicação de padrões de projetos. Além disso, com esse trabalho foi possível aplicar alguns dos conceitos definidos pela engenharia de software como: a importância de documentar, de ter domínio da engenharia reversa de software, e por fim testar o software. Vale também salientar os conceitos de redes de computadores, e de programação (orientação a objetos) que fez com que fosse possível compreender o OpenICE a nível de código. Todo esse arcabouço de conhecimento foi essencial para a realização desse trabalho, evidenciando assim a importância de se ter uma boa base acadêmica para a realização de tais pesquisas.

Como objetivos futuros fica a total integração do HAM a interface de usuário do OpenICE, bem como sua integração com qualquer outra ferramenta existente na suíte do OpenICE. Também existe espaço para a construção de uma documentação mais detalhada do funcionamento do OpenICE.

REFERÊNCIAS

HIMSS, What is interoperability. Disponível em: <http://www.himss.org/library/interoperability-standards/what-is-interoperability>. Acesso em: 27 em Agosto de 2016

CLASSE e OLVEIRA, Trabalhando com engenharia reversa – Revista engenharia de software Magazine 59. Disponível em: <http://www.devmedia.com.br/trabalhando-com-engenharia-reversa-revista-engenharia-de-software-magazine-59/28203>. Acesso em: 27 em Agosto de 2016

MDPNP, OpenICE. Disponível em: <https://github.com/mdpnp/mdpnp>. Acesso em: 27 em Agosto de 2016

Team, RTI.Getting Started Manual. Disponível em: < https://community.rti.com/static/documentation/connext-dds/5.2.3/doc/manuals/connext_dds/RTI_ConnextDDS_CoreLibraries_GettingStarted.pdf> Acesso em: 17 de Agosto 2016.

Team, RTI. User's Manual. Disponível em: < https://community.rti.com/static/documentation/connext-dds/5.2.3/doc/manuals/connext_dds/RTI_ConnextDDS_CoreLibraries_UsersManual.pdf >. Acesso em: 17 de Agosto 2016.

MDPNP, OpenICE. Disponível em: <https://www.openice.info>. Acesso em: 27 em Agosto de 2016

SAPAGE, Engenharia reversa. Disponível em:

<http://www2.ic.uff.br/~otton/graduacao/informaticaI/apresentacoes/eng_reversa.pdf >.

Acesso em: 12 de Dezembro 2016

GOLDMAN, Medical Connectivity for improving Safety and Efficiency. Disponivel em:<

http://www.mdpnp.org/uploads/ASA_May_2006_Newsletter_on_MD_PnP.pdf>. Acesso em:

12 de Dezembro 2016

STEFANELLI et al., Engenharia Reversa: Discusão sobre validade e legalidade desta prática. Disponivel em: <http://www.stefanelli.eng.br/webpage/a-engenharia-reversa.html>. Acesso em: 12 de Dezembro 2016

APÊNDICE A – IDL do HAM

```
typedef string<64> SystemID;

typedef string<64> Model;

typedef string<16> Specializations;

typedef sequence<string, 32> UnitIdentifierList;

typedef sequence<string, 32> UnitSymbolList;

typedef sequence<string, 32> ValueMeasurementList;

typedef sequence<string, 32> DateList;

typedef sequence<string, 32> TimeList;

typedef string<64> UniqueDeviceIdentifier;

struct HAM_Device {
   SystemID unique_system_id; //@key
   VendorMetricIdentifier vendor_metric_id; //@key
   Model device_model;
   Specializations specialization_code;
   UnitIdentifierList unit_id; //@key
   UnitSymbolList unit_symbol;
   ValueMeasurementList values;
   DateList measurement_date;
   TimeList measurement_time;
}; //@top-level true //@Extensibility MUTABLE_EXTENSIBILITY
#pragma keylist Device unique_system_id vendor_metric_id device_model
specialization_code unit_id unit_symbol
const string HAM_DeviceTopic = "HAM_Device"
```

APÊNDICE B – Testes realizados

Teste aferidor de pressão:

```
#include <QCoreApplication>
#include <gtest/gtest.h>
#include <interface.h>

Interface* inter = new Interface();
//loadingXmlTests
TEST(xmlInputTest, xmlInputed){

    ASSERT_TRUE(inter->xmlInput("/home/alisson/atributosBloodPressure.xml","/home/alisson/medicaoBloodPressure.xml"));

}

//attribuits tests
TEST(getVendorMetricIdTest, OMRON_HEALTHCARE){//Following the model:EXPECT_EQ(expected, actual)

    EXPECT_EQ(QString("OMRON HEALTHCARE"), inter->attr->getVendorMetricId());
}
TEST(getModelTest, Model){

    EXPECT_EQ(QString("HEM-7081-IT"), inter->attr->getModel());
}
TEST(getSystemIDTest, ID){

    EXPECT_EQ(QString("00220922580803CC"), inter->attr->getSystemID());
}
TEST(getSystemIDTest, Specialization){
```

```
    EXPECT_EQ(QString("1007"), inter->attr->getSpecializationCode());
}
//measurement tests
TEST(getUnitTest, ListOfUnits){

    EXPECT_EQ(QStringList()<<"3872"<<"2720"<<"512", inter->mesur->getUnitList());
}
TEST(getMeasurementTest, ListOfMesureament){

    EXPECT_EQ(QStringList()<<"(111.000000, 65.000000, 80.000000)"<<"60.000000"<<"0.000000", inter->mesur->getMeasurementList());
}
TEST(getDateListTest, ListDates){

    EXPECT_EQ(QStringList()<<"09.03.2016"<<"09.03.2016"<<"09.03.2016", inter->mesur->getDateList());
}
TEST(getTimeListTest, ListOfTime){


EXPECT_EQ(QStringList()<<"07:25:30.000"<<"07:25:30.000"<<"07:25:30.000", inter->mesur->getTimeList());
}
TEST(getSymbolListTest, ListOfSymbols){

    EXPECT_EQ(QStringList()<<"mmHg"<<"bpm"<<"", inter->mesur->getSymbolList());
}

int main(int argc, char *argv[])
{
    testing::InitGoogleTest(&argc, argv);
    QCoreApplication a(argc, argv);
```

```
    return RUN_ALL_TESTS();//calling all tests
    return a.exec();
}
```

Teste da balança:

```
#include <QCoreApplication>
#include <gtest/gtest.h>
#include <interface.h>

Interface* inter = new Interface();
//loadingXmlTests
TEST(xmlInputTest, xmlInputed){

    ASSERT_TRUE(inter->xmlInput("/home/alisson/atributosWeightScale.xml","/home/alisson/medicaoWeightScale.xml"));

}

//attribuites tests
TEST(getVendorMetricIdTest, OMRON_HEALTHCARE){//Following the model:EXPECT_EQ(expected, actual)

    EXPECT_EQ(QString("OMRON HEALTHCARE"), inter->attr->getVendorMetricId());
}
TEST(getModelTest, Model){

    EXPECT_EQ(QString("HBF-206IT"), inter->attr->getModel());
}
TEST(getSystemIDTest, ID){
```

```
    EXPECT_EQ(QString("002209225807E86B"), inter->attr->getSystemID());
}
TEST(getSystemIDTest, Specialization){

    EXPECT_EQ(QString("100f"), inter->attr->getSpecializationCode());
}
//measurement tests
TEST(getUnitTest, ListOfUnits){


EXPECT_EQ(QStringList()<<"1731"<<"1297"<<"1952"<<"544"<<"6784"<<"544"<<"2
368"<<"544"<<"512"<<"512", inter->mesur->getUnitList());
}
TEST(getMeasurementTest, ListOfMesureament){


EXPECT_EQ(QStringList()<<"4.500000"<<"159.000000"<<"0.000000"<<"0.000000"
<<"0.000000"<<"0.000000"<<"0.000000"<<"(0.000000, 0.000000, 0.000000,
0.000000)"<<"0.000000"<<"(0.000000, 25.000000)", inter->mesur-
>getMeasurementList());
}
TEST(getDateListTest, ListDates){


EXPECT_EQ(QStringList()<<"09.01.2017"<<"09.01.2017"<<"09.01.2017"<<"09.01.2
017"<<"09.01.2017"<<"09.01.2017"<<"09.01.2017"<<"09.01.2017"<<"09.01.2017"<<
"09.01.2017", inter->mesur->getDateList());
}
TEST(getTimeListTest, ListOfTime){


EXPECT_EQ(QStringList()<<"04:27:00.000"<<"04:27:00.000"<<"04:27:00.000"<<"04
:27:00.000"<<"04:27:00.000"<<"04:27:00.000"<<"04:27:00.000"<<"04:27:00.000"<<"
04:27:00.000"<<"04:27:00.000", inter->mesur->getTimeList());
```

```
}
TEST(getSymbolListTest, ListOfSymbols){

    EXPECT_EQ(QStringList()<<"kg"<<"cm"<<"kg m-
2"<<"%"<<"cal"<<"%"<<"y"<<"%"<<""<<"", inter->mesur->getSymbolList());
}

int main(int argc, char *argv[])
{
    testing::InitGoogleTest(&argc, argv);
    QCoreApplication a(argc, argv);

    return RUN_ALL_TESTS();//calling all tests
    return a.exec();
}
```

Teste do simulador de oxímetro

```
#include <QCoreApplication>
#include <gtest/gtest.h>
#include <interface.h>

Interface* inter = new Interface();
//loadingXmlTests
TEST(xmlInputTest, xmlInputed){

    ASSERT_TRUE(inter-
>xmlInput("/home/alisson/atributosOximeter.xml","/home/alisson/medicaoOximeter.x
ml"));

}

//attribuites tests
```

```
TEST(getVendorMetricIdTest, OMRON_HEALTHCARE){//Following the
model:EXPECT_EQ(expected, actual)

    EXPECT_EQ(QString(""), inter->attr->getVendorMetricId());
}
TEST(getModelTest, Model){

    EXPECT_EQ(QString(""), inter->attr->getModel());
}
TEST(getSystemIDTest, ID){

    EXPECT_EQ(QString("1133557799BBDDFF"), inter->attr->getSystemID());
}
TEST(getSystemIDTest, Specialization){

    EXPECT_EQ(QString(""), inter->attr->getSpecializationCode());
}
//measurement tests
TEST(getUnitTest, ListOfUnits){

    EXPECT_EQ(QStringList()<<"544"<<"2720", inter->mesur->getUnitList());
}
TEST(getMeasurementTest, ListOfMesureament){

    EXPECT_EQ(QStringList()<<"96.500000"<<"63.500000", inter->mesur-
>getMeasurementList());
}
TEST(getDateListTest, ListDates){

    EXPECT_EQ(QStringList(), inter->mesur->getDateList());
}
TEST(getTimeListTest, ListOfTime){

    EXPECT_EQ(QStringList(), inter->mesur->getTimeList());
```

```
}
TEST(getSymbolListTest, ListOfSymbols){

    EXPECT_EQ(QStringList()<<"%"<<"bpm", inter->mesur->getSymbolList());
}

int main(int argc, char *argv[])
{
    testing::InitGoogleTest(&argc, argv);
    QCoreApplication a(argc, argv);

    return RUN_ALL_TESTS();//calling all tests
    return a.exec();
}
```

Teste do subscriber em Java:

```
package test;

import static org.junit.Assert.*;

import java.util.ArrayList;
import java.util.Arrays;

import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import org.mdpnp.apps.testapp.IceAppsContainer;


public class GettingDataTest {
        @Before
        public void setUp() throws Exception {
```

```
}

public void testGetDataPressure() {

        ice.HAM_Device a;
        a = new ice.HAM_Device();
        a.vendor_metric_id = "OMRON HEALTHCARE";
        a.specialization_code = "1007";
        a.unique_system_id = "00220922580803CC";
        ArrayList<String> unitList = new ArrayList<String>();
        unitList.add("3872");
        unitList.add("2720");
        unitList.add("512");
        a.unit_id.userData.addAll(unitList);
        ArrayList<String> valueList = new ArrayList<String>();
        valueList.add("(111.000000, 65.000000, 80.000000)");
        valueList.add("60.000000");
        valueList.add("0.000000");
        a.values.userData.addAll(valueList);
        a.device_model = "HEM-7081-IT";
        ArrayList<String> dateList = new ArrayList<String>();
        dateList.add("09.03.2016");
        dateList.add("09.03.2016");
        dateList.add("09.03.2016");
        a.measurement_date.userData.addAll(dateList);
        ArrayList<String> timeList = new ArrayList<String>();
        timeList.add("07:25:30.000");
        timeList.add("07:25:30.000");
        timeList.add("07:25:30.000");
        a.measurement_time.userData.add(timeList);
        ArrayList<String> symbolList = new ArrayList<String>();
        symbolList.add("mmHg");
        symbolList.add("bpm");
        symbolList.add("");
```

```
        a.unit_symbol.userData.addAll(symbolList);
        Assert.assertEquals(a, IceAppsContainer.getData());
}
@Test
public void testGetDataWeight() {

        ice.HAM_Device a;
        a = new ice.HAM_Device();
        a.vendor_metric_id = "OMRON HEALTHCARE";
        a.specialization_code = "100f";
        a.unique_system_id = "002209225807E86B";
        a.device_model = "HBF-206IT";
        ArrayList<String> unitList = new ArrayList<String>();
        unitList.add("1731");
        unitList.add("1297");
        unitList.add("1952");
        unitList.add("544");
        unitList.add("6784");
        unitList.add("544");
        unitList.add("2368");
        unitList.add("544");
        unitList.add("512");
        unitList.add("512");
        a.unit_id.userData.addAll(unitList);
        ArrayList<String> valueList = new ArrayList<String>();
        valueList.add("4.500000");
        valueList.add("159.000000");
        valueList.add("0.000000");
        valueList.add("0.000000");
        valueList.add("0.000000");
        valueList.add("0.000000");
        valueList.add("0.000000");
        valueList.add("(0.000000, 0.000000, 0.000000, 0.000000)");
        valueList.add("0.000000");
```

```
valueList.add("(0.000000, 25.000000)");
a.values.userData.addAll(valueList);

ArrayList<String> dateList = new ArrayList<String>();
dateList.add("09.01.2017");
dateList.add("09.01.2017");
dateList.add("09.01.2017");
dateList.add("09.01.2017");
dateList.add("09.01.2017");
dateList.add("09.01.2017");
dateList.add("09.01.2017");
dateList.add("09.01.2017");
dateList.add("09.01.2017");
dateList.add("09.01.2017");
a.measurement_date.userData.addAll(dateList);
ArrayList<String> timeList = new ArrayList<String>();
timeList.add("04:27:00.000");
timeList.add("04:27:00.000");
timeList.add("04:27:00.000");
timeList.add("04:27:00.000");
timeList.add("04:27:00.000");
timeList.add("04:27:00.000");
timeList.add("04:27:00.000");
timeList.add("04:27:00.000");
timeList.add("04:27:00.000");
timeList.add("04:27:00.000");
a.measurement_time.userData.add(timeList);
ArrayList<String> symbolList = new ArrayList<String>();
symbolList.add("kg");
symbolList.add("cm");
symbolList.add("kg m-2");
symbolList.add("%");
symbolList.add("cal");
symbolList.add("%");
```

```
            symbolList.add("y");
            symbolList.add("%");
            symbolList.add("");
            symbolList.add("");
            a.unit_symbol.userData.addAll(symbolList);
        }
        @Test
        public void testGetDataOxi() {

            ice.HAM_Device a;
            a = new ice.HAM_Device();
            a.specialization_code = "1133557799BBDDFF";
            ArrayList<String> unitList = new ArrayList<String>();
            unitList.add("544");
            unitList.add("2720");
            a.unit_id.userData.addAll(unitList);
            ArrayList<String> valueList = new ArrayList<String>();
            valueList.add("96.500000");
            valueList.add("63.500000");
            a.values.userData.addAll(valueList);
            a.device_model = "";
            Assert.assertEquals(a, IceAppsContainer.getData());
        }

}
```