INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Sistema Multi-plaforma segura de conectividade para hardware e software médico

NUNO HENRIQUE SOARES DOS REIS ALMEIDA
Outubro de 2017

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Secure Multi-Platform Connectivity Solution for Health Hardware and Software

Nuno Henrique Soares dos Reis Almeida

Dissertação para obtenção do Grau de Mestre em Engenharia Informática

Orientador: Filipe de Faria Pacheco Pacheco

Porto, Outubro 2017

Dedicatória

A toda a minha família que me suportou até ao fim

- Nuno Almeida

Agradecimentos

A toda a minha família que me suportou até ao fim

- Nuno Almeida

Índice

Lista de Figuras

Glossário

ASCII	*American Standard Code for Information Interchange*
Cloud	Armazenamento e Processamento distribuído na internet
Dump	Descarregamento total de dados
Endpoint	Recetor final
HL7	*Health Level Seven*
IoT	*Internet of Things*
Input	Entrada
IVD	*In-Vitro Diagnostic*
MLLP	*MLLP - Minimal Lower Layer Protocol*
Output	Saída
Reverse Engineer	Processo de reversão de sistemas para descoberta do conteúdo
SBC	*Single-Board Computer*
Sockets	Sistema de comunicação entre computadores
WebAPI	Protocolo de comunicação
Workflow	Fluxo de trabalho

1 Introdução

Neste capítulo introdutório irá ser explicado de forma simples a dissertação e o projeto desenvolvido, referindo brevemente a motivação e os objetivos da mesma.

Será também dado a conhecer um pouco da organização em que esta dissertação foi desenvolvida, nomeadamente a FUJIFILM, e os contributos deste trabalho.

1.1 Âmbito

Esta dissertação foi realizada no âmbito do Mestrado de Engenharia Informática para conclusão do mesmo.

Este projeto foi elaborado em colaboração com a FUJIFILM Europe GmbH, empresa esta que financiou e geriu algumas variáveis na criação do produto descrito neste documento.

1.2 FUJIFILM

A FUJIFILM é uma empresa japonesa fundada em 1934 que começou por produzir filme fotográfico prosseguindo com uma expansão para filmes de raios X, máquinas fotográficas,...

Os produtos da FUJIFILM têm variados papéis em diferentes ambientes e áreas: desde a utilização das camaras para fotografia profissional, às camaras industriais e máquinas de raios X utilizadas na saúde.

Desde a sua criação, a FUJIFILM ampliou as suas operações para outros países, como o Brasil, os Estados Unidos da América (FUJIFILM U.S.A., Inc.), Alemanha (FUJIFILM Europe GmbH), China, …

A FUJIFILM é vista como uma empresa maioritariamente de venda de artigos relacionados com a fotografia – amadora e profissional. No entanto, hoje pode-se dizer que o foco de trabalhos – por exemplo, no 3º quadrimestre do ano de 2016 apenas 10% da empresa consistia na secção de fotografia – já a secção de saúde tem o valor de 17% (Figura 1-1). Isto deve-se a descida drástica da necessidade de utilização de filmes fotográficos a partir do ano 2001 (Figura 1-2).

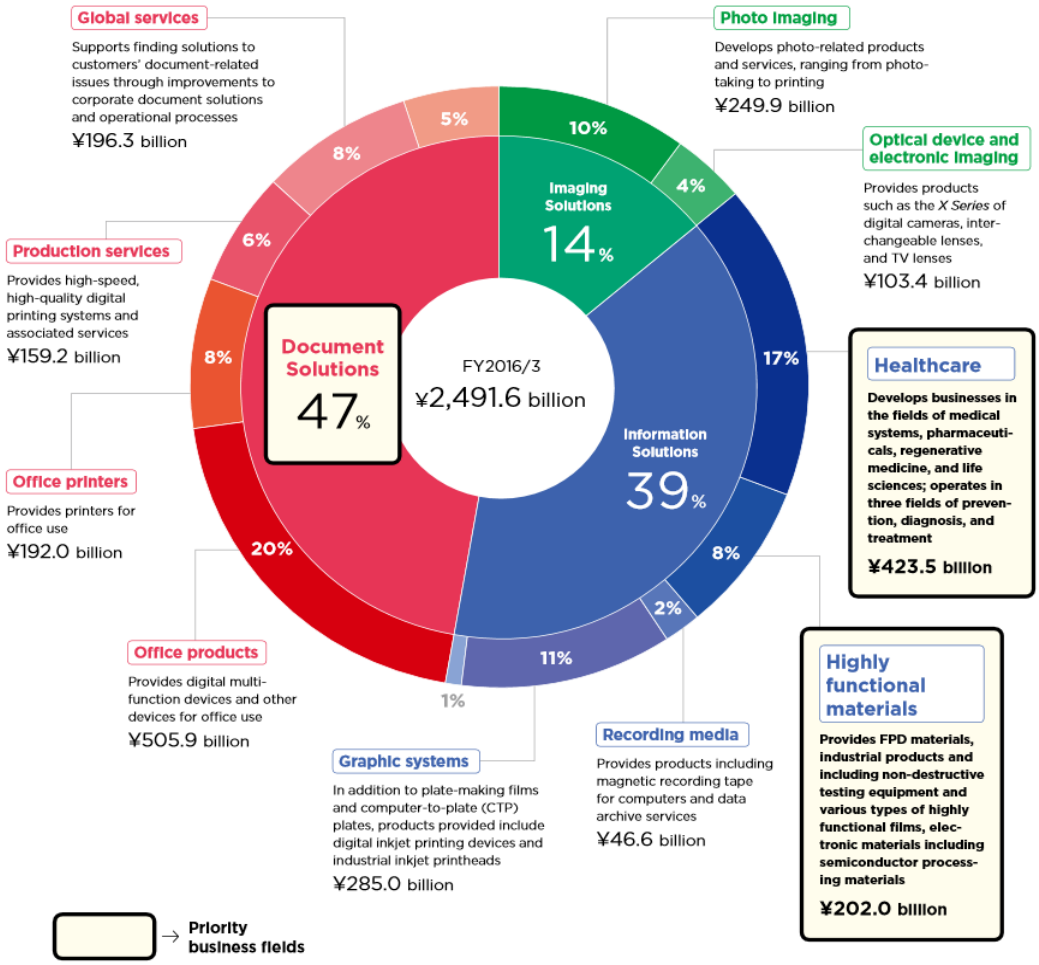

Figura 1-1 – Segmentos de Mercado da FUJIFILM no 3º Quadrimestre de 2016 (FUJIFILM Holdings Corporation, 2016)

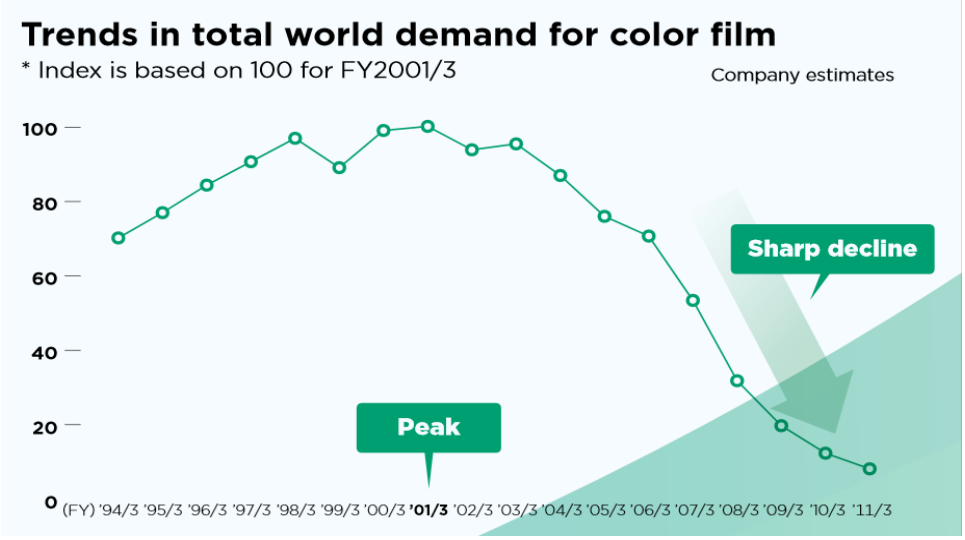

Figura 1-2 – Procura mundial de filme fotográfico
(FUJIFILM Holdings Corporation, 2016)

1.3 Contexto e Motivação

O departamento de saúde e medicina da FUJIFILM tem presentemente uma enorme quantidade de diferentes sistemas em produção, sendo a maioria destes não inteligentes – não tem capacidade de comunicação com outros alvos que não o utilizador local, isto é – apenas permitem a saída de dados via impressão em papel ou exibidos num pequeno ecrã na máquina.

Presentemente existe uma procura por sistemas inteligentes que, para além da comunicação local com o médico, também se liguem com sistemas hospitalares já instalados nos vários locais e transfira os dados de maneira coesa e consistente para tais, proporcionando centralização de dados sobre todos os exames efetuados pela máquina, com a possibilidade de associação com a ficha do paciente alvo do exame permitindo assim uma mais eficaz visualização do histórico do paciente para exames futuros e consequentemente a diminuição de erros pela parte do médico devido a falta de conhecimento do histórico do paciente. Tais sistemas inteligentes produzem também dados valiosos para a administração do hospital ou centro de saúde, potencializando a criação de estatísticas de uso de consumíveis, tempos de manutenção de equipamento, entre outros.

1.4 Objetivos

O projeto referente a esta dissertação deverá solucionar o problema descrito na secção 1.3 Contexto e Motivação, de forma a satisfazer o mercado em relação aos sistemas médicos Fujifilm (atuais e futuros), no âmbito da sua conexão e interligação com outros sistemas hospitalares. Pode-se definir então os seguintes objetivos gerais:

1. Investigação do modo de trabalho dos médicos nos sistemas de saúde FUJIFILM.
2. Estudo sobre tecnologias existentes que facilitem a elaboração do sistema.
3. Desenho de um sistema que ajude ou solucione o problema da falta de comunicação dos sistemas FUJIFILM.
4. Criação de um protótipo com um número reduzido de funcionalidades.
5. Avaliação do protótipo com base no fluxo atual dos médicos.

2 Ambiente

Neste capítulo será descrito o ambiente em que este projeto foi desenvolvido, explicando as várias tecnologias atuais envolvidas e modo de utilização das mesmas.

2.1 Contexto Ambiental

2.1.1 Sistemas IVD FUJIFILM

Sistemas IVD (*In-Vitro Diagnostic Medical Devices*) são sistemas para diagnóstico, monitorização ou teste de compatibilidade para uso médico (Global Harmonization Task Force, 2007). Estes sistemas são comuns em hospitais pois fornecem rapidamente resultados de um exame com uma precisão aceitável ou até ótima.

A FUJIFILM criou e vende alguns sistemas IVD:

FUJIFILM AG1: Sistema IVD com o objetivo de diagnóstico do vírus da gripe (*Influenza type A/B*). Esta máquina utiliza técnicas de amplificação utilizando prata, de forma a gerar resultados em tempo útil (inferior a 1 minuto e meio) (Mikinaga Mori, 2012) (Figura 2-1).

Figura 2-1 – FUJIFILM AG1 (Intermedical - FUJIFILM AG1)

FUJIFILM NX500: Sistema IVD com o objetivo de testar vários parâmetros de química clinica, contendo 29 tipos de testes num espaço de tempo entre 1 a 6 minutos, dependente do teste realizado (FUJIFILM NX500 Brochure, p. 2) (Figura 2-2).

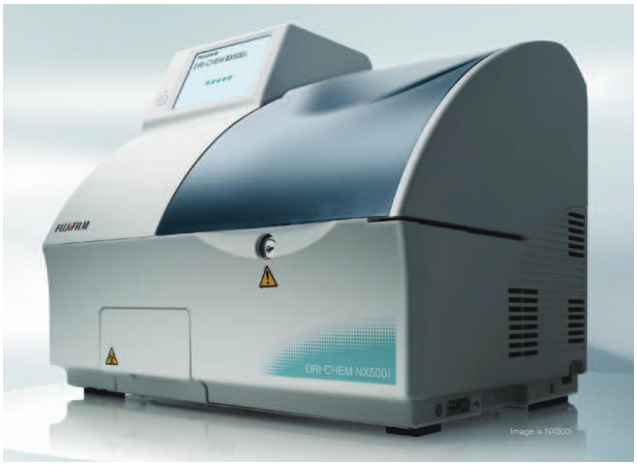

Figura 2-2 – FUJIFILM NX500 (FUJIFILM NX500 Brochure, p. 1)

FUJIFILM AU10: Sistema IVD para uso veterinário, com o objetivo de testar doenças de fígado e endócrino (Figura 2-3).

Figura 2-3 – FUJIFILM AU10 (Immunodiagnostics Testing now available from Fujifilm, 2014)

2.1.2 Modo de utilização

Neste momento, os médicos utilizam estes sistemas IVD de forma ineficiente (**Erro! A origem a referência não foi encontrada.**). O médico começa por recolher a amostra do paciente e insere-a na máquina, juntamente com os reagentes necessários. O médico agora fica à espera do resultado que é impresso e mostrado no ecrã da máquina, recolhe-o e analisa-o, descartando o papel quando deixa de ser necessário.

Este método é ineficiente e pode levar a inconsistência de dados nos LIS/HIS do hospital, caso o médico adicione os resultados à mão.

Existe também a possibilidade dos dados nem sequer serem registados e tornar-se assim num exame "fantasma".

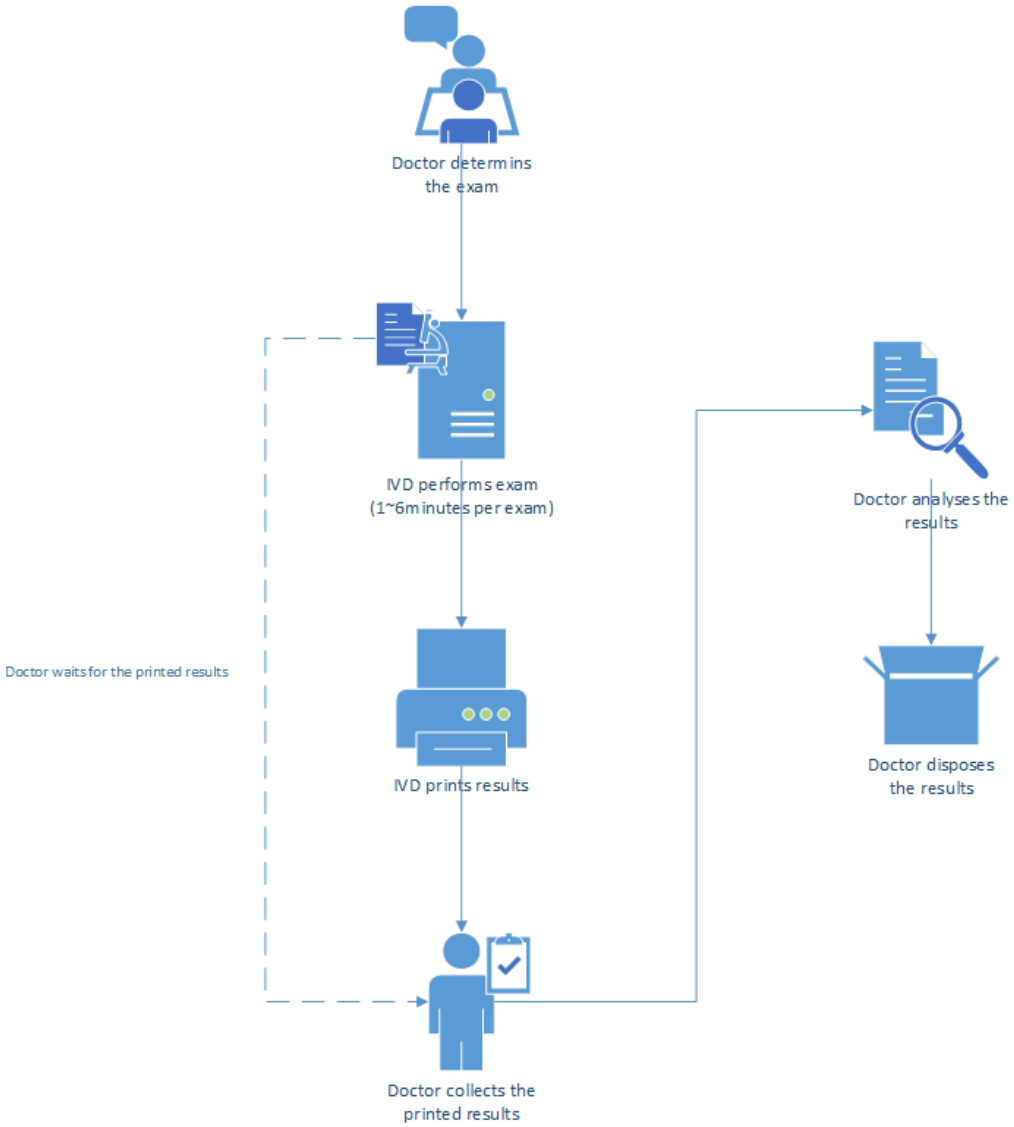

Figura 2-4 – Fluxo atual de uso de sistemas IVD

2.2 Análise de Valor

Em termos de identificação de oportunidade, pode-se dizer que se verificou a existência da necessidade da interligação dos sistemas IVD com os sistemas hospitalares de armazenamento de dados. Isto leva a falta de consistência no fluxo de diagnóstico médico pois os dados não estão coesos e completos.

Após uma análise a vários possíveis clientes, determinou-se que existem poucos sistemas que façam esta ligação de forma simples, potencializando assim o controlo total do buraco no mercado para estes sistemas.

Para a capitalização deste mercado, definiu-se uma forma de criar um sistema de interligação de vários componentes hospitalares que satisfaça a necessidade dos clientes.

Seguindo o processo de brainstorming do sistema, as várias possibilidades foram analisadas em vários parâmetros de forma a escolher a melhor – análise do custo de desenvolvimento, análise do custo final, vantagens e desvantagens gerais das ideias.

Após esta seleção, foi criado um *design* estático com as especificações de funcionalidades e tecnologias necessárias ao desenvolvimento do projeto.

Em termos de valor, a solução proposta trás inúmeros benefícios para o cliente. Perante o produto, este poderá:

- Gerar diagnósticos de forma consistente
- Reduzir a frustração na configuração de sistemas de interligação
- Aumentar a coesão de dados nos sistemas hospitalares
- Aumentar a produtividade dos médicos

Todos estes benefícios apenas contribuem para um pequeno sacrifício:

- O custo do produto

Propõe-se então a criação de uma máquina de interligação de sistemas IVD, de fácil configuração e baixo custo de modo a aumentar a produtividade dos clientes e a unir os dados das várias competências médicas.

Em termos do modelo canvas (Figura 2-5), este negócio pode ser apresentado da seguinte forma:

- Parcerias principais – RaspberryPi e Samsung, pois são os principais fornecedores de hardware que iremos necessitar,
- Actividades-Chave – Investigação e desenvolvimento, Prototipagem e Venda do Produto e Serviços,
- Recursos Principais – A equipa de desenvolvimento e a equipa de vendas (da parte da FujiFilm Portugal e FujiFilm Europe),
- Propostas de Valor – A criação do sistema de conectividade para máquinas "burras" (máquinas IVD), juntamente com a sua interligação com os sistemas hospitalares e a criação de serviços de gestão e estatística,
- Relacionamente com Clientes – será feito principalmente pelas equipas contratadas pela FujiFilm, no entanto iremos fornecer um site de feedback,
- Canais – A distribuição será feita pela FujiFilm Portugal e FujiFilm Europe,
- Segmentos de Clientes – Segmento Hospitalar e Veterinário, isto é, gestores hospitalares, ministério da saúde, hospitais veterinários, …
- Estrutura de Custos – Os custos atribuídos serão maioritariamente na parte de Desenvolvimento de Software e na compra de Máquina de prototipagem e produção,
- Fontes de Receita – Venda das máquinas de conectividade e serviços associados.

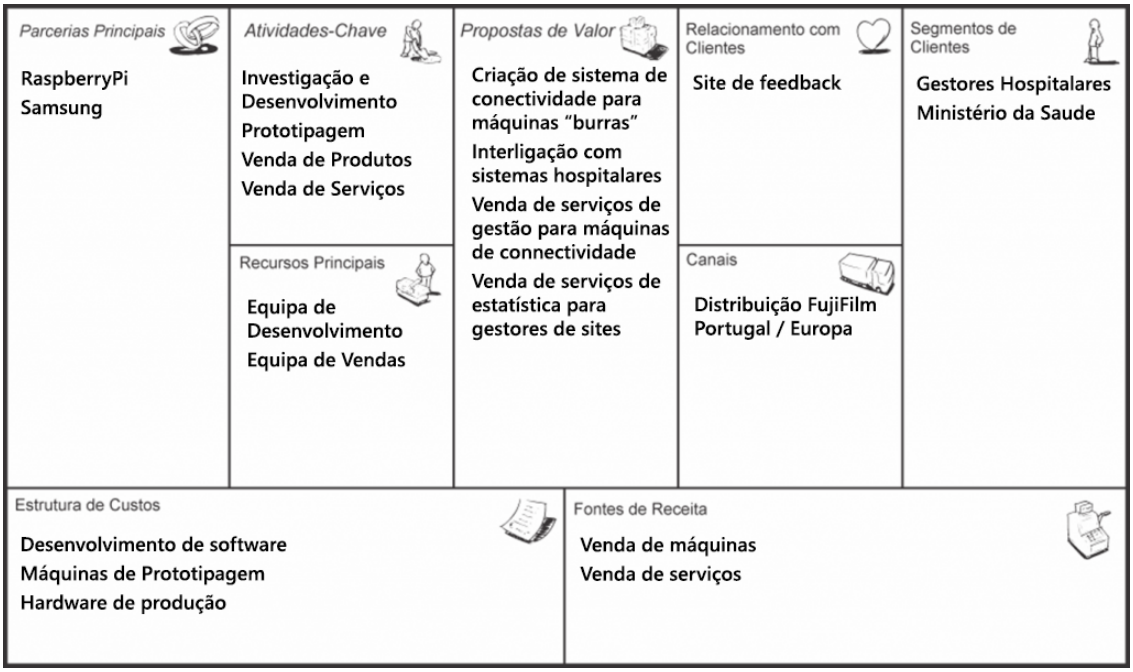

Figura 2-5 – Modelo Canvas

2.3 Produtos Relacionados

Os produtores de máquinas IVD ainda não adotam a maioria dos *standards* de comunicação e usam protocolos proprietários que dificultam a integração destes com outros sistemas.

Neste ambiente, existem de momento poucas empresas com capacidades para a elaboração de um sistema de conectividade. De momento existe:

- *Conworx* – é uma empresa que foi recentemente comprada pela Siemens (Siemens, 2016) que criou o sistema *POCcelerator* (Figura 2-6) e o *UniPOC* (Figura 2-7).
    - O *POCcelerator* agrega as informações fornecidas pelas máquinas IVD e disponibiliza-as em forma de uma página web ou em forma de uma aplicação local.
    - O *UniPOC* gere todas as máquinas IVD no local, com possibilidade de configuração destas remotamente.

Figura 2-6 – *Conworx POCcelerator*

Figura 2-7 – *Conworx UniPOC*

- FUJIFILM ChemXPort (Figura 2-8) – A FUJIFILM desenvolveu também um *software* para a agregação de dados enviados pelos sistemas IVD FUJIFILM. O sistema desenhado nesta dissertação irá substituir este sistema, acrescentando valor e funcionalidades aos utilizadores atuais do ChemXPort.

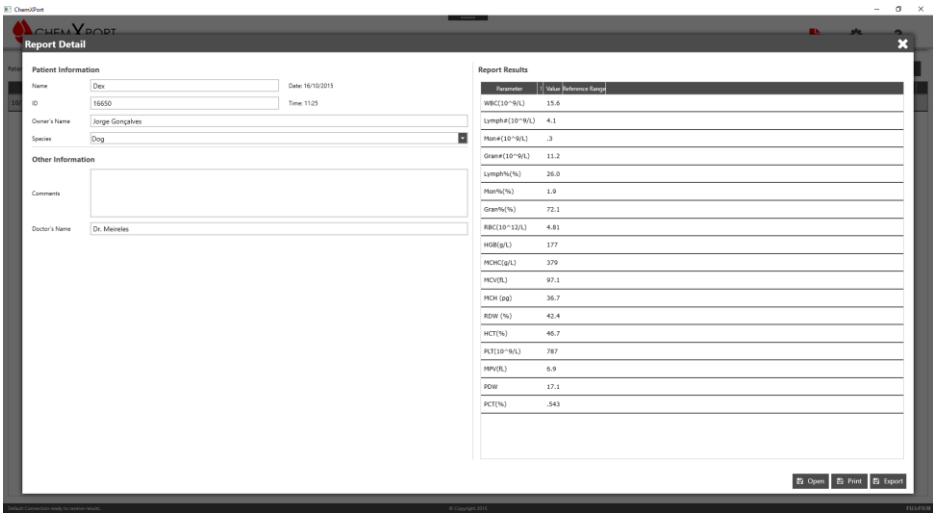

Figura 2-8 – FUJIFILM ChemXPort

- *Radiometer Aqure Enterprise* – É outro *software* de interligação com máquinas IVD, com o mesmo nível de funcionalidades do *Conworx POCcelerator* (Figura 2-9).

Figura 2-9 – *Radiometer Aqure Enterprise* (AQURE Enterprise – open, smart, integrated)

2.4 Ideologias e Metodologias

2.4.1 Internet of Things (IoT)

A "Internet das Coisas" é uma visão ideológica sobre o que nos rodeia como potencial forma de intercomunicação tecnológica – isto é – a partir da captura de dados, processamento e comunicação, a "Internet das Coisas" cria serviços para estes dados captados sejam consumidos por outras aplicações ou "coisas".

Uma forma de mais fácil compreensão desta ideologia é a comunicação de todos os sistemas entre eles, tornando estes mais eficientes e "inteligentes".

Os sistemas *Internet of Things* são utilizados em vários tipos de aplicações, como transportes, e-saúde, casas inteligentes, … (ITU-T - Telecommunication standerization sector of ITU, 2012, p. 4).

As características principais do IoT são:

- Interconexão,
- Serviços relacionados com "coisas" (no mundo físico com associação do mundo virtual),
- Heterogeneidade,
- Dinamicidade,
- Grande escala.

(ITU-T - Telecommunication standerization sector of ITU, 2012, p. 5)

2.5 Tecnologias

2.5.1 Hardware

2.5.1.1 Single-Board Computer

Um *Single-Board Computer* (SBC) é um computador constituído inteiramente por uma placa que funciona sem necessidade da utilização de ranhuras de expansão.

Estas placas têm a vantagem de serem de tamanho reduzido, consumirem pouca energia e de terem um preço muito baixo – mantendo ainda assim um poder de processamento considerável (mas inferior) face a um computador normal.

Desde a popularização do conceito de *Internet of Things*, o número de modelos deste tipo de placas tem vindo a aumentar drasticamente, permitindo o desenvolvimento rápido de aplicações que fazem interface com sensores e outros sistemas.

Alguns fabricantes de SBCs permitem a revenda dos mesmos com modificações feitas pelo revendedor e existem também SBCs concebidos para produção massiva. Por outro lado, a maioria dos Single-Board Computers no mercado são apenas para prototipagem e desenvolvimento próprio, com a revenda proibida.

Alguns exemplos destas placas são:

- Raspberry Pi (Figura 2-10)– a placa de desenvolvimento mais conhecida do mercado.
- Odroid C1/C2/XU4 (Figura 2-11),
- Pine64,
- BananaPi,
- OrangePi,
- LattePanda,
- MinnowBoard MAX,
- DragonBoard 410c,
- Asus Tinkerboard.

Figura 2-10 – Raspberry PI

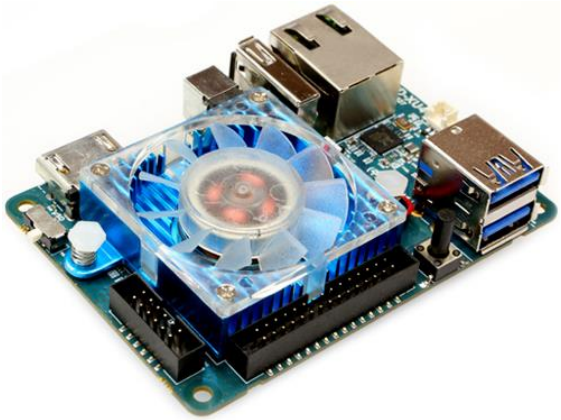

Figura 2-11 – Odroid XU4

2.5.2 Software

2.5.2.1 Bases de Dados

***Relational Database* (SQL)**

Uma base de dados relacional é um sistema de armazenamento de dados generalista que segue a organização proposta por E. F. Codd – organização esta que segue um esquema de tuplos com relações entre dados com várias ligações – ligações unárias, binárias, … (Codd, 1970).

Atualmente, esta organização é mais reconhecida por linhas e colunas, na forma de uma tabela. Também é normalmente associada à utilização da linguagem SQL para a execução de pesquisas ou modificações nas várias tabelas.

Alguns exemplos de sistemas de bases de dados relacionais são:

- MySQL (e MySQL Cluster),

- Oracle,
- MSSQL,
- MariaDB.

NoSQL

NoSQL é também um tipo de base de dados, no entanto, este não segue regras completamente estruturadas, isto é, existe a possibilidade de guardar dados dinâmicos em termos de forma e estrutura.

Dentro do género de bases de dados existem vários tipos como:

- Key-Value Store,
- Document Store,
- Wide Column Store (Column family store).

(Mimi Gentz, 2016)

Em termos de implementações de NoSQL existem por exemplo:

- Cassandra,
- Couchbase,
- MongoDB,
- HBase,
- …

2.5.2.2 HL7

Overview

HL7 (Health Level-7) é um *standard* que define o tipo de mensagens enviado por equipamentos e *software* que lida com informação relacionada com saúde, de forma a interoperabilidade deste sistemas ser assegurada.

Este *standard* apresenta várias versões, sendo as subversões 2.x.x as mais utilizadas atualmente (Corepoint Health, 2009). O primeiro *standard* da versão 2 publicado data o ano 1989 (Health Level Seven, 2017).

Neste pequeno sumário do *standard* irá estar em foco a versão 2.

Sistema de mensagens

O sistema de mensagens definido no HL7 é um sistema simples de fácil implementação. Este consiste em dois tipos de funções principais normalmente encontrados nas comunicações de rede: o sistema de envio e o sistema de receção.

O sistema de envio gera as mensagens de acordo com o *standard* e envia-as para o recetor. Este, para uma receção com sucesso necessita de "imediatamente" (Seven, HL7: Version 2 Standard, 2010) enviar a confirmação da chegada da mensagem.

Tabela 2-1 – Exemplo de uma mensagem HL7 (Ringhold Whitepaper, 2007)

```
MSH|^~\&|GHH LAB|ELAB-3|GHH OE|BLDG4|200202150930||ORU^R01|CNTRL-3456|P|2.4
PID|||555-44-4444||EVERYWOMAN^EVE^E^^^^L|JONES|19620320|F|||153
FERNWOOD . . . . . .
OBR|1|845439^GHH OE|1045813^GHH
LAB|15545^GLUCOSE|||200202150730|||||||| . . . . . .
OBX|1|SN|1554-5^GLUCOSE^POST 12H CFST:MCNC:PT:SER/PLAS:QN||^182|mg/dl|70_105|H|||F
```

No exemplo na Tabela 2-1 – Exemplo de uma mensagem HL7 , pode-se ver uma mensagem HL7 de versão 2. Para compreender esta mensagem é necessário ter a noção de segmentos, campos e componentes.

A mensagem é composta por segmentos. Estes podem ser identificados pelo fim de linha. Dentro destes segmentos existem os campos – estes são delimitados pelos carateres "|" – e dentro destes campos existem os componentes – delimitados pelos carateres "^".

Todas as mensagens começam também por "MSH", seguidas pela descrição do tipo da mensagem – este exemplo é uma mensagem ORU (*Observation Result*) – visto no campo 9 do segmento 1.

Existem vários tipos de mensagens definidos pelo *standard*. Os mais comuns são:

- ACK – Mensagem de confirmação (*Acknowledgement*)

- ADT – Mensagem de admissão, dispensa ou transferência (*Admint Discharge Transfer*)

- ORM – Mensagem de "encomenda" (*Order*)

- ORU – Mensagem de resultado (Observation Result)

Sistema de comunicação

Para além da definição da mensagem em si, o HL7 também define o sistema de comunicação usado para a transferência de mensagens – o MLLP (Minimal Lower Layer Protocol) (Health Level Seven, 2003).

Este sistema deve ser implementado sobre TCP/IP e deverá ter o seguinte formato:

1. Começar com um caracter especial para início do bloco – normalmente o carater ASCII referente ao número hexadecimal 0x0b.

2. Dados da mensagem HL7

3. Um carater especial para finalizar o bloco – normalmente usado o carater ASCII referente ao número hexadecimal 0x1c.

4. Finalizada por um fim de linha – sendo padrão a utilização do carater ASCII 0x0d.

3 Avaliação Generalista

3.1 Avaliação do Produto

O desenho do sistema deve ter em consideração o risco existente na modificação de dados médicos, podendo qualquer erro afetar o diagnóstico pela parte do médico. Devido a isto, o sistema terá de fazer o mínimo de modificações aos resultados possíveis, e aquando a modificação de tais, terá de ser extremamente preciso e previsível.

O sistema terá de ser eficiente em termos de tempo, isto é, deverá ser rápido o suficiente para o médico não notar demora no processamento e transferência dos dados. Este sistema também terá de ser eficiente em termos de tempo, no sentido de ajudar o médico utilizar o seu tempo de melhor forma.

A solução terá de ser de fácil configuração para os administradores de sistemas, de forma a aumentar a satisfação dos mesmos.

Qualquer tipo de aplicação ou dados terá de ser segura, isto é, tanto como a obtenção ilícita de dados como a obtenção ilícita de código fonte deverá ser minimizada ao máximo.

3.2 Avaliação de Tecnologias Existentes

Em termos de avaliação de tecnologias existentes, serão apenas focadas as tecnologias com possível utilização na solução.

Estas serão avaliadas pelos seguintes parâmetros:

- Usabilidade e conhecimento atual pela parte da equipa de desenvolvimento,
- Velocidade,
- Segurança,
- Estabilidade.

4 Soluções

4.1 Solução Geral – Modificação do modo de utilização de sistemas IVD

Como descrito na secção 2.1.2 - Modo de utilização, o médico deve manter-se perto da máquina aquando a criação de um diagnóstico, e depois inserir os dados manualmente do exame num computador.

Assim, é proposto a modificação do modo de utilização dos sistemas IVD, onde o médico apenas faz o exame na máquina, podendo deixar esta a processar e consequentemente utilizar o tempo extra na pesquisa de histórico do paciente.

Aquando do término do exame feito pela máquina, o médico receberia a informação do diagnóstico no LIS/HIS, sem necessidade de se deslocar e sem necessidade de inserir manualmente dados (Figura 4-1 – Modo de utilização proposto).

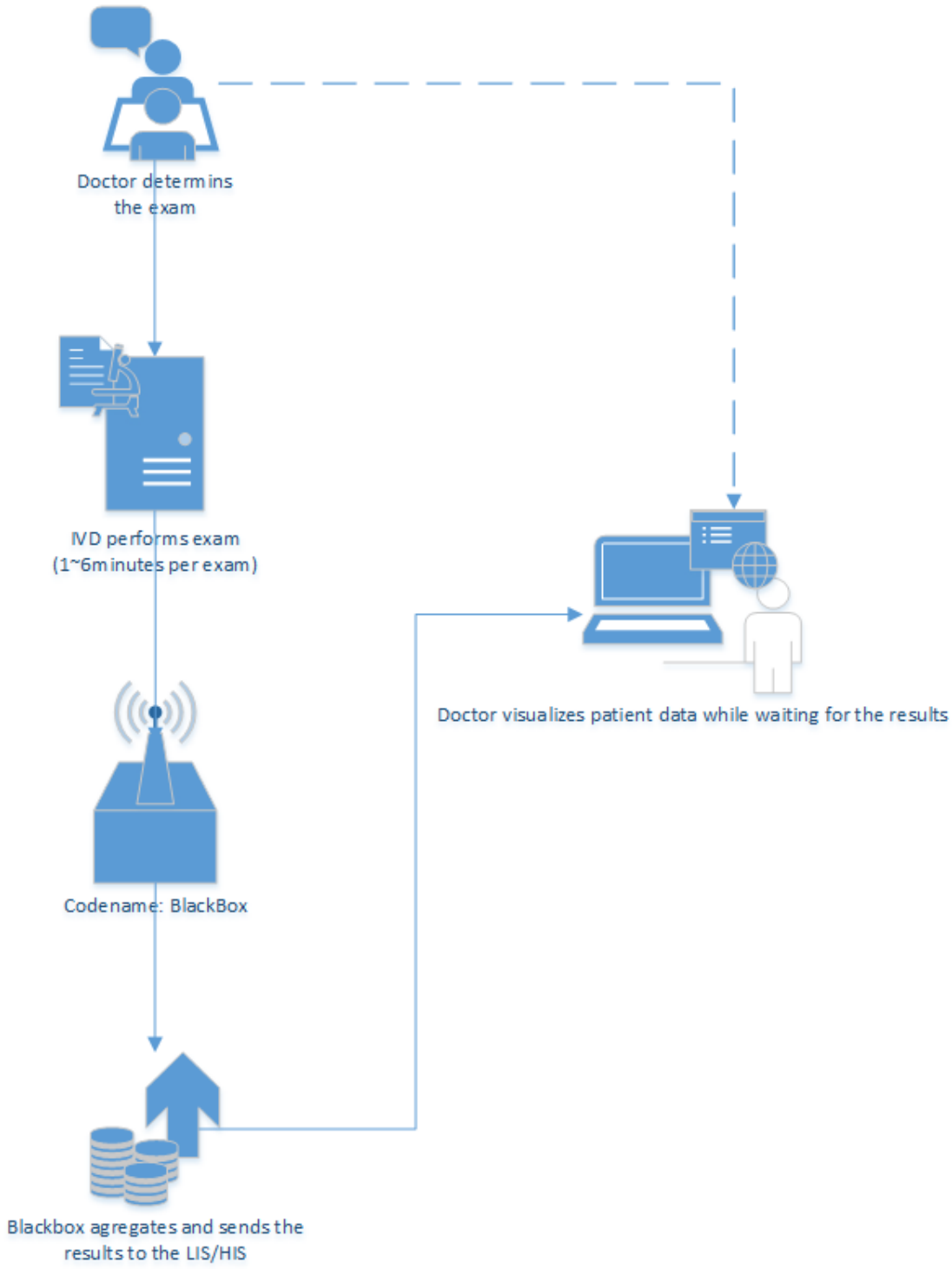

Figura 4-1 – Modo de utilização proposto

4.2 Solução Técnica

Perante a solução de *workflow* descrita na secção 4.1 – Solução Geral – Modificação do modo de utilização de sistemas IVD, existem várias possibilidades para a implementação técnica de tal sistema, passando por:

- A criação de uma aplicação baseada apenas em *software* ou adaptação da aplicação FUJIFILM ChemXPort.
- A utilização de microcontroladores especializados para o processamento das tramas enviadas pelas máquinas IVD.
- A criação de um sistema híbrido, baseado em ambos *hardware* e *software*.

Estas três possibilidades serão descritas nos seguintes subcapítulos.

4.2.1 Possibilidade 1 – Solução de Software Pura

A primeira solução seria a criação de uma aplicação implementada em *software* apenas, isto é, com a utilização de sistemas de bases de dados e a implementação de uma aplicação de conexão com as máquinas IVD permitiria armazenar temporariamente ou permanentemente os resultados gerados pela máquina.

Os dados armazenados poderiam então ser transferidos para o sistema de dados hospitalar (LIS/HIS) via aplicações de integração de sistemas de saúde como o *Mirth* e o *Rhapsody* ou implementado uma forma de comunicação estandardizada com um protocolo também este *standard*, como o HL7. Potencialmente, este sistema também poderia ser integrado com a FUJIFILM Cloud para acesso remoto dos dados (Figura 4-2).

No caso de existir um cliente sem necessidade de manter os dados num servidor – tal como clínicas de reduzida dimensão – os dados podem ficar guardados apenas localmente no computador em que a aplicação se encontra (Figura 4-3).

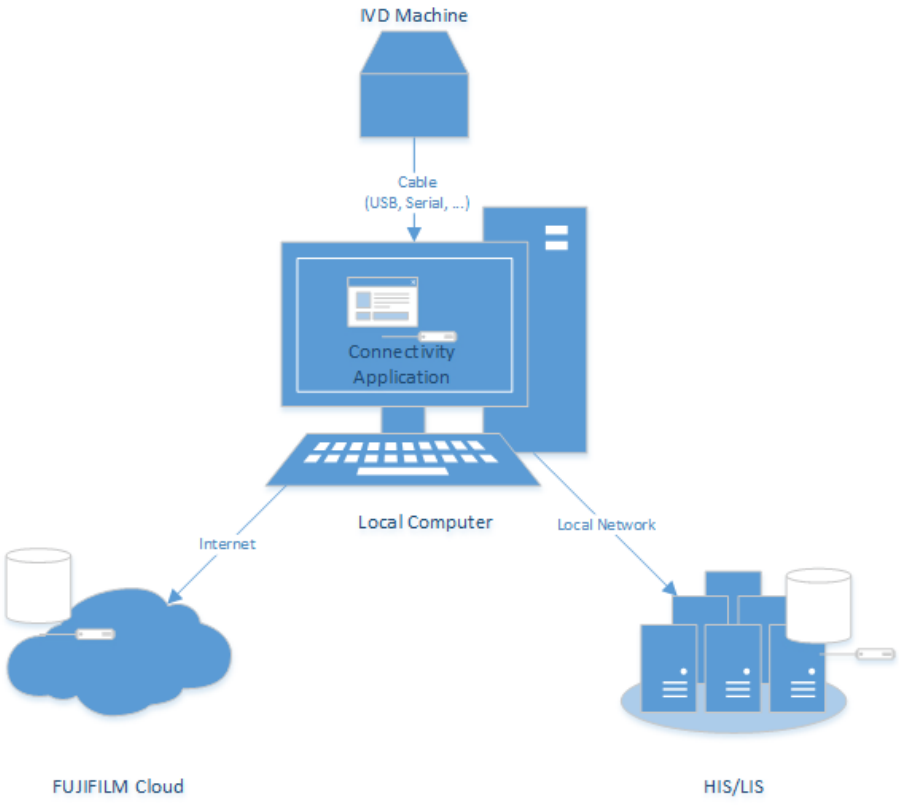

Figura 4-2 – Possível arquitetura de uma aplicação com conexão a sistemas exteriores.

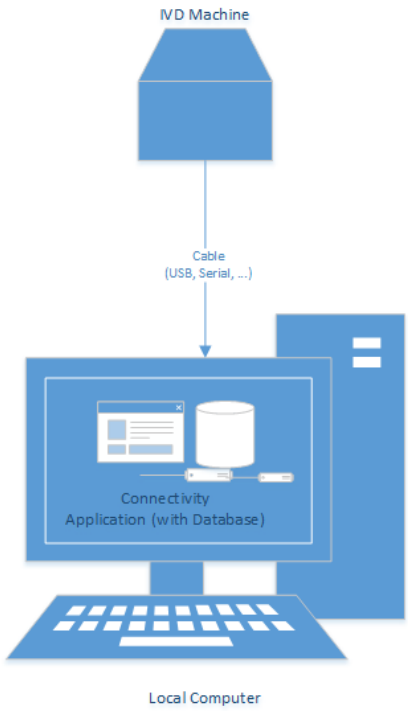

Figura 4-3 – Possível arquitetura de uma aplicação sem necessidade de comunicação com o exterior.

4.2.2 Possibilidade 2 – Solução de Hardware Pura

Esta solução consiste no desenho de um microcontrolador especializado para descodificar as tramas recebidas.

Este microcontrolador então transmitia os dados num protocolo *standard* para uma placa de rede, para retransmitir para o local final de armazenamento de dados (Figura 4-4).

O armazenador de dados teria de ter uma aplicação que entendesse tal protocolo. A maioria dos sistemas hospitalares é capaz da descodificação dos dados encapsulados no protocolo HL7, assim seria uma questão apenas da escolha do tipo de conexão (sockets, WebApi, …).

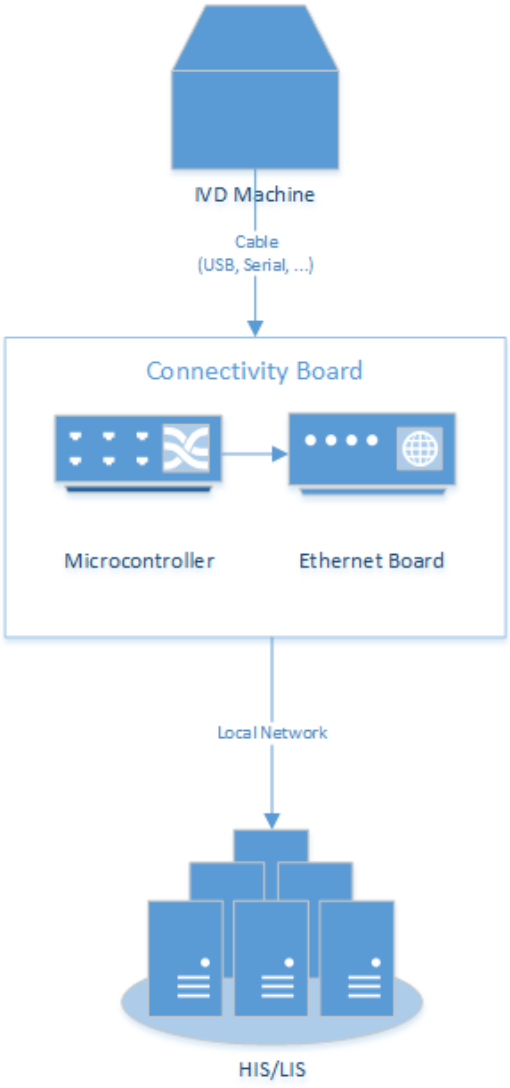

Figura 4-4 – Possível arquitetura de um sistema baseado em hardware.

4.2.3 Possibilidade 3 – Solução Híbrida (Software e Hardware)

A terceira possibilidade utiliza ideias de ambas as soluções anteriores, criando assim uma solução híbrida.

Em vez de ligar diretamente a um computador local, a máquina IVD é ligada a um pequeno computador/placa, que por sua vez é ligado aos armazenadores de dados finais (Figura 4-5).

Esta pequena máquina iria ter uma aplicação com algoritmos descodificadores das tramas recebidas, transformando os dados num protocolo *standard* (HL7) e transmitindo o resultado para o *endpoint* desejado.

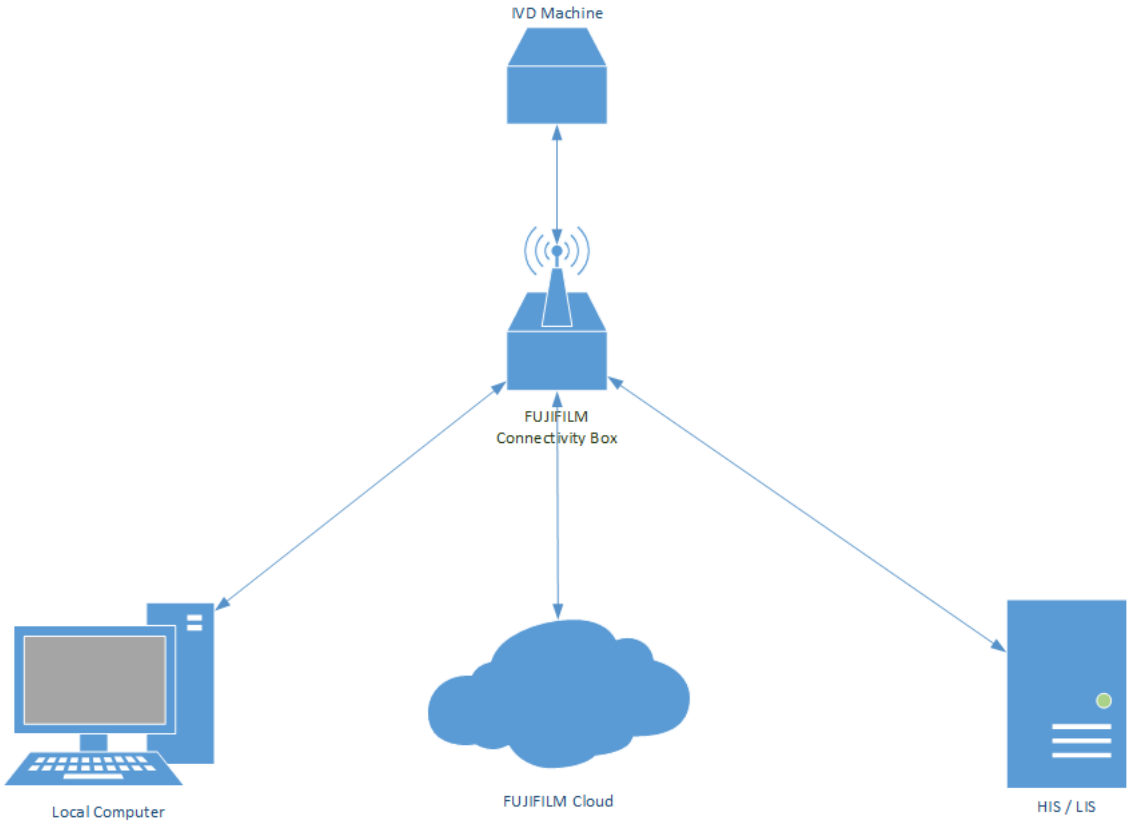

Figura 4-5 – Possível arquitetura de um sistema híbrido.

4.2.4 Comparação e Solução Final

Possibilidade 2

A solução de *hardware* puro foi descartada imediatamente devido à incapacidade de adaptação a várias máquinas IVD.

Para esta suportar vários tipos de tramas enviados pelas máquinas IVD, o *design* do microcontrolador teria de refletir algoritmos de vários tipos, com inteligência suficiente em reconhecer o tipo de dados que está a receber.

Poderia ser desenhado um microcontrolador específico para cada tipo de IVD, no entanto, isto traria custos enormes à FUJIFILM em termos de desenho, prototipagem e teste destes.

Existe também o problema de a equipa de desenvolvimento ser focada em *Software* – seria necessário a subcontratação de uma empresa para desenhar os microcontroladores.

Apesar destas falhas, a possibilidade 2 traria alguns benefícios, como elevada segurança – para o *reverse engineer* do algoritmo de descodificação das tramas, seria necessário a desassemblagem do microcontrolador e a investigação cuidadosa dos circuitos – tarefa que, para além de trabalhosa, necessita de *hardware* especializado para tal.

Possibilidade 1 e 3

Ambas estas soluções potencialmente resolvem o problema exposto – no entanto, a possibilidade 3 foi a escolhida.

As soluções 1 e 3 iriam ambas permitir a consolidação dos resultados dos diagnósticos das máquinas IVD, permitindo a ligação à FUJIFILM Cloud e ao sistema local HIS/LIS, utilizando protocolos de fácil configuração para o sistema local.

Estas possibilidades diferem, no entanto, na quantidade de desenvolvimento necessário – a possibilidade 1 tem a vantagem neste ponto.

A solução 1 poderia adaptar o sistema FUJIFILM ChemXPort e adicionar conectividade a este. Isto significaria que os algoritmos de descodificação das tramas já estariam implementados e testados num ambiente real.

Já a solução 3 tem a vantagem de ser uma camada intermédia controlada completamente pela FUJIFILM, possibilitando a estabilização do sistema pois deixa de existir o problema da modificação do *software* de terceiros, como *Windows Updates*.

O facto de esta camada ser controlada pela FUJIFILM trás também outros benefícios, como a mais fácil gestão do licenciamento e vendas de máquinas, trazendo maiores dificuldades na pirataria do sistema (é necessário o *dump* dos dados dentro da placa e depois a tentativa da de-ofuscação dos programas).

Trás também a possibilidade de modificação de *hardware*, caso seja necessário um *input* específico de certa máquina ou o mais fácil diagnóstico de um problema com uma peça física.

Após a enumeração dos prós e contras, as vantagens da solução 3 foram designadas como mais fortes que as vantagens da solução 1.

5 Connectivity Box

Neste capítulo será explicada a implementação, não só da Box de *hardware*, mas como as aplicações e interfaces necessárias para o funcionamento da solução.

5.1 Design

O processo de *design* da solução foi levado por várias iterações, começando num modelo simples, mas tornando-se sucessivamente mais complexo à medida que a necessidade de funcionalidades ia surgindo ou novas restrições foram aplicadas.

Estas novas funcionalidades foram levantadas pela equipa de gestão da FUJIFILM ou por médicos que estão em contacto com a equipa de desenvolvimento.

Iteração 1

A primeira iteração consistia na utilização de uma Connectivity Box com ligações a múltiplas máquinas IVD (Figura 5-1).

Esta *box*, em termos de ingestão de dados, teria uma aplicação tradutora das tramas enviadas pelas máquinas. Esta aplicação inseria os dados numa base de dados relacional local de forma a armazena-los permanentemente. Seria também disponibilizado uma página web na *box* para a consulta dos resultados armazenados.

Esta solução foi rapidamente ultrapassada devido à falta de flexibilidade e a restrições da mesma:

- Seria necessário a *box* ter uma grande capacidade de armazenamento para guardar todos os dados.
- O médico para o acesso aos dados teria de se ligar a cada uma das *box* de forma diferente. Uma tentativa da resolução deste problema foi a interconexão entre as *box,* no entanto esta ideia foi posta de parte pelo *management* (Figura 5-2).
- Não existia coesão nos dados. A ideia proposta na Figura 5-2 potencialmente resolveria este problema.

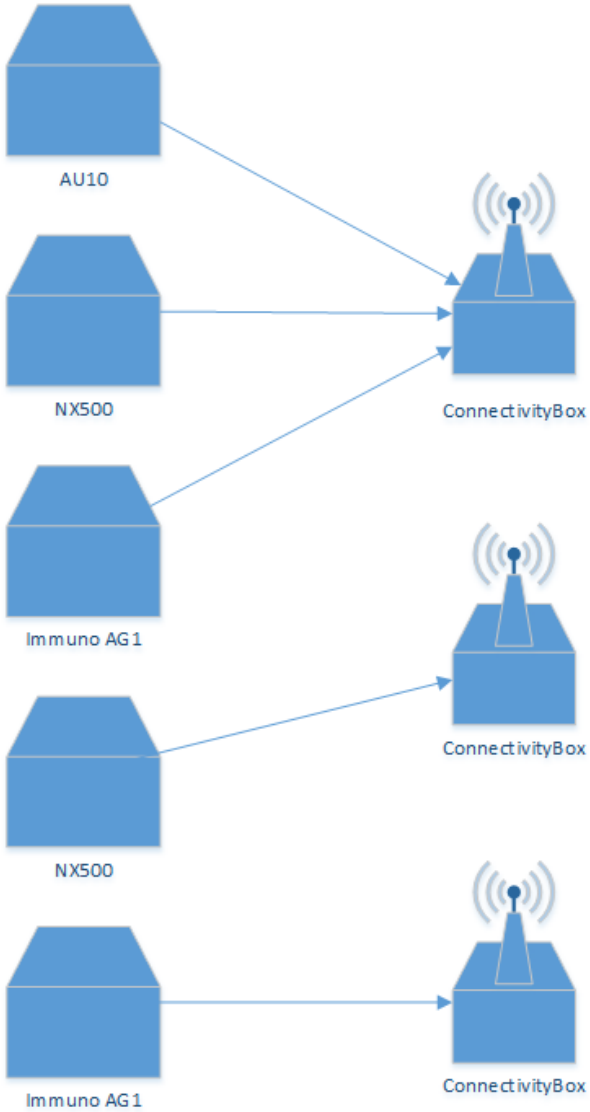

Figura 5-1 – Design Arquitetural – Iteração 1 – Simples

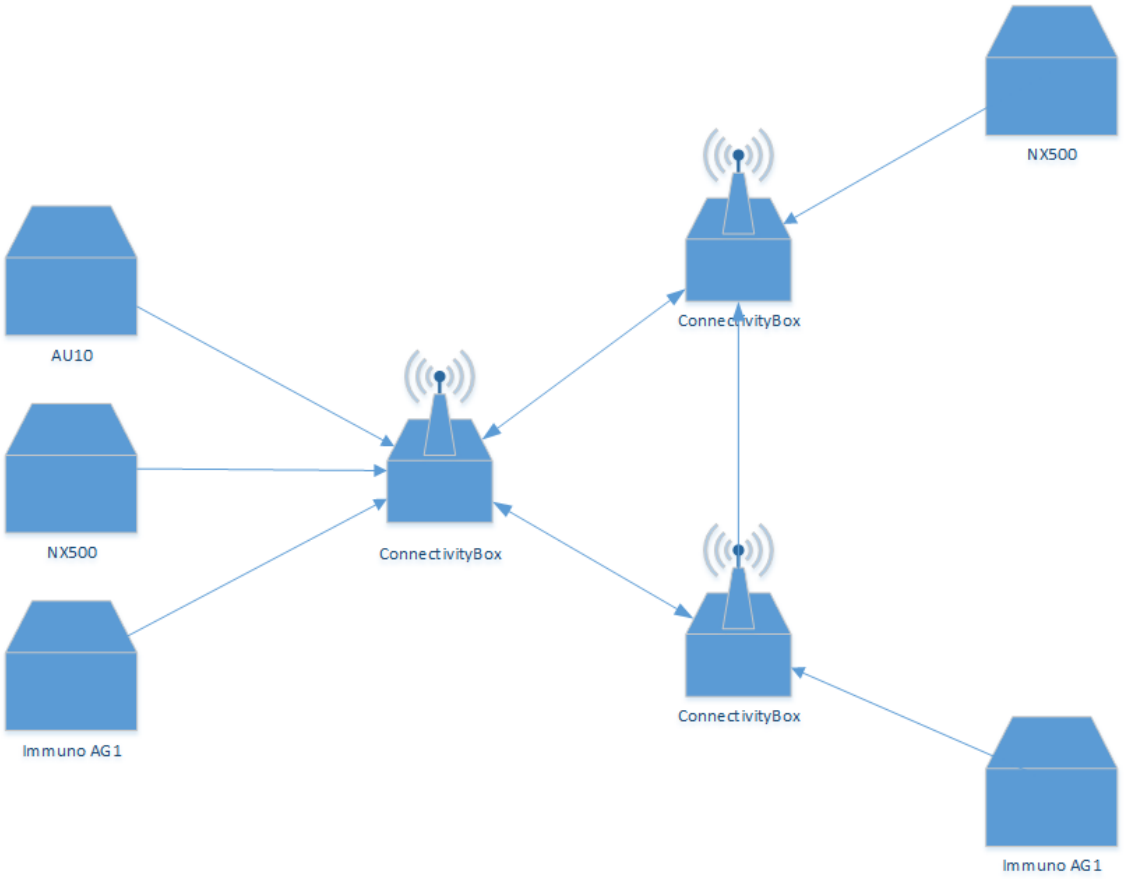

Figura 5-2 – Design Arquitetural – Iteração 1 – Intercomunicação

Iteração 2

A segunda iteração foi elaborada com o objetivo de resolver os problemas da iteração 1.

As múltiplas ligações às máquinas IVD foram mantidas, no entanto, de forma criar a coesão de dados, foi adicionado um servidor local a qual todas as *box* têm acesso (Figura 5-3). Desta forma, o problema de armazenamento de dados fica também resolvido, pois armazenamento para servidores é barato e eficaz.

Por esta altura, o management também exprimiu a intenção de fornecer serviços Cloud para clientes interessados. Definiu-se então que a Cloud iria ser um agregado de réplicas dos servidores locais, de forma a ter todos os dados necessários à consulta remotamente (Figura 5-4).

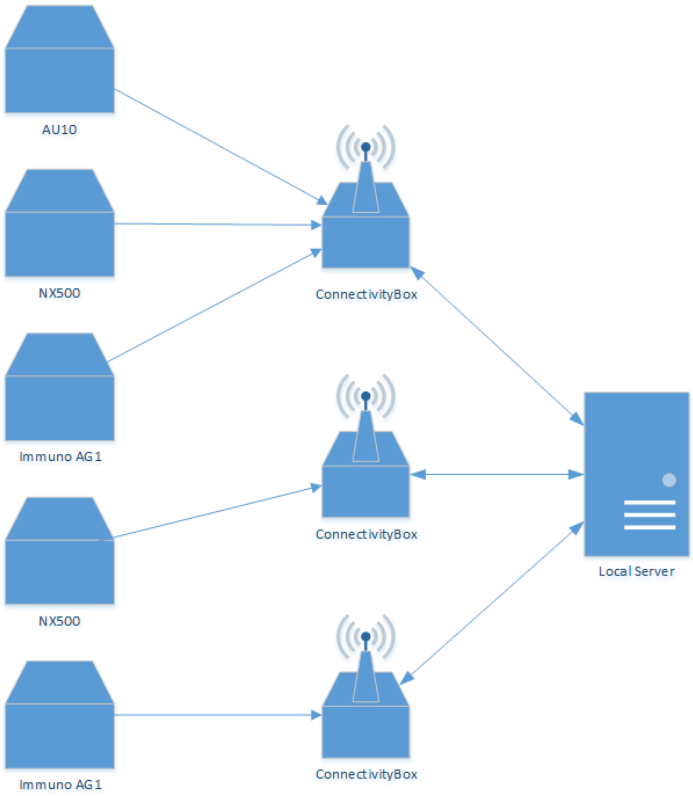

Figura 5-3 – Design Arquitetural – Iteração 2 – Simples

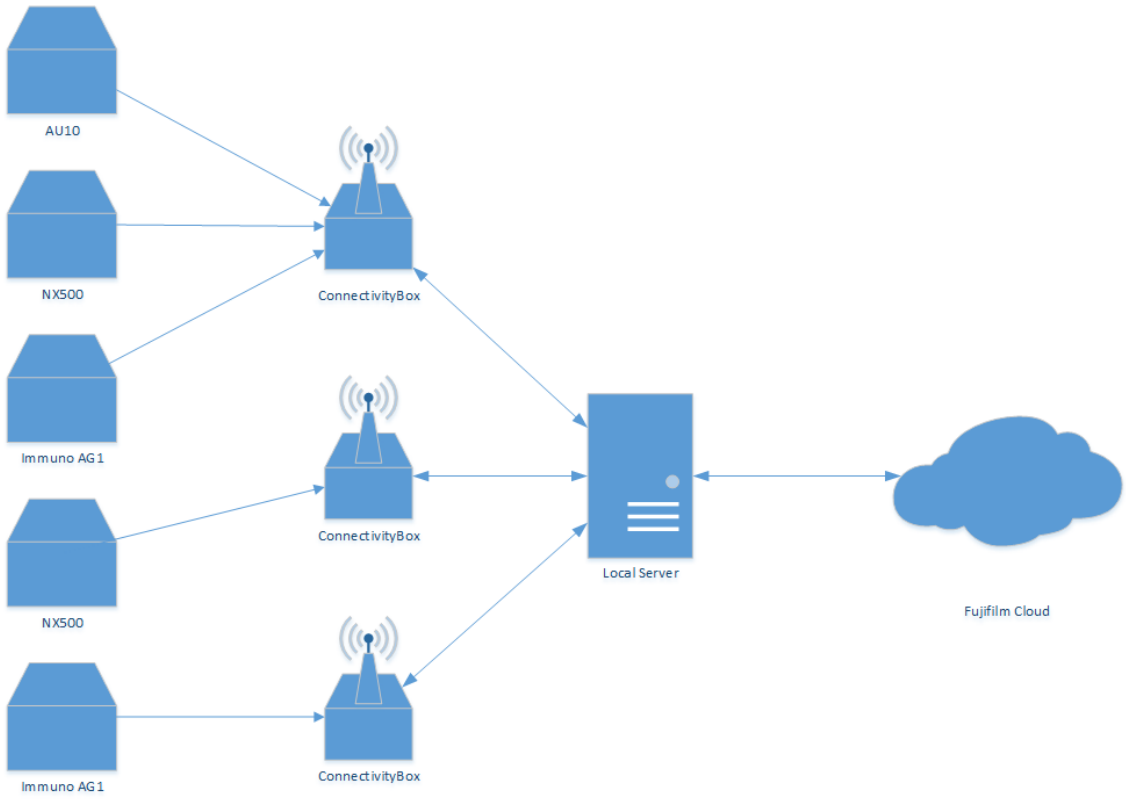

Figura 5-4 – Design Arquitetural – Iteração 2 – Serviços Cloud

Iteração 3

A terceira iteração começou quando foi definido pelo *management* que cada *box* deveria apenas ligar a uma máquina IVD.

Para esta iteração, foram apresentados 2 designs – o primeiro foi o desenho elaborado na iteração 2, mas com capacidade para a solução funcionar sem um servidor local, isto é, as box ligam diretamente à Cloud, além da ligação do servidor local. O segundo desenho incorpora o conceito da *MiniBox* – uma *box* mais pequena que se ligava às máquinas IVD em vez da ConnectivityBox, comunicando com esta via rede sem fios.

A MiniBox iria utilizar a ideologia do IoT – uma pequena caixa que mede o ambiente a volta dela, ligado a uma um sistema principal que iria depositar os dados na cloud ou um servidor local onde possam ser processados.

Esta *MiniBox* possibilitaria a modularização do inventário destas – isto é – cada *MiniBox* poderia ser vendida como módulos diferentes (um módulo para porta série, um modulo para porta USB, …). No entanto, traria a desvantagem da complexidade acrescida no Marketing e na implementação do sistema – o que levou a que este design fosse posto de lado.

Decidiu-se então pelo *design* da Figura 5-5.

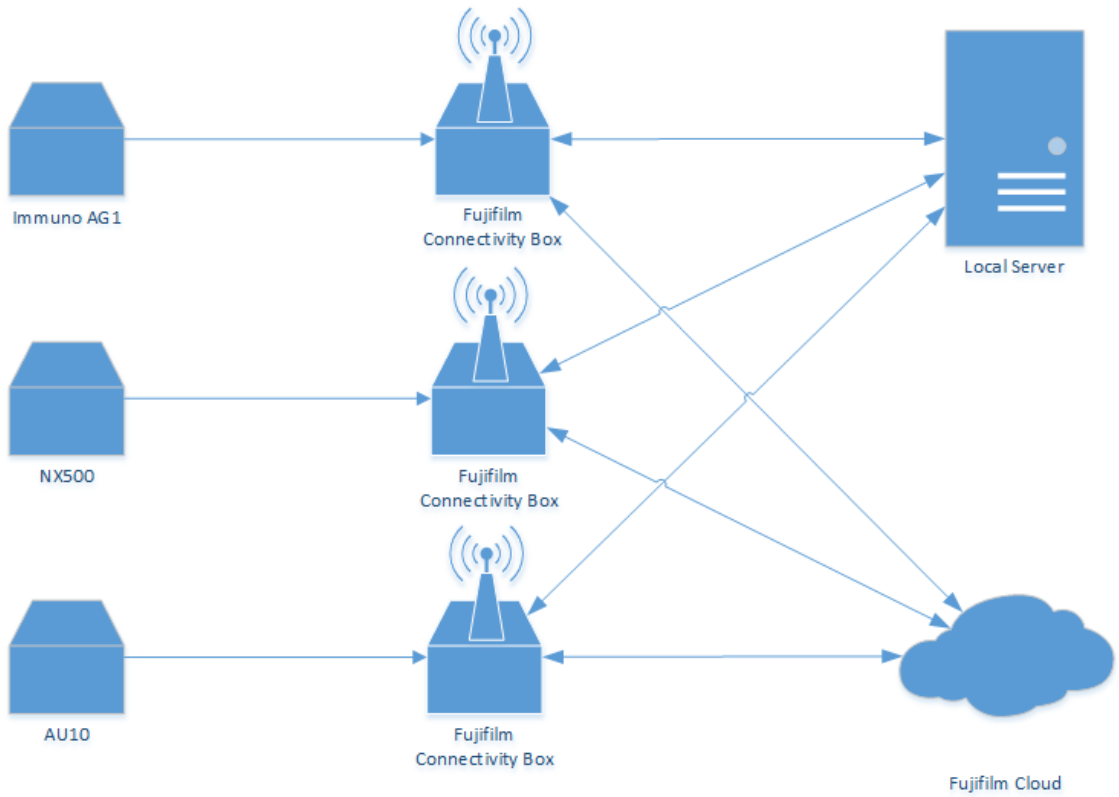

Figura 5-5 – Design Arquitetural – Iteração 3 - Simples

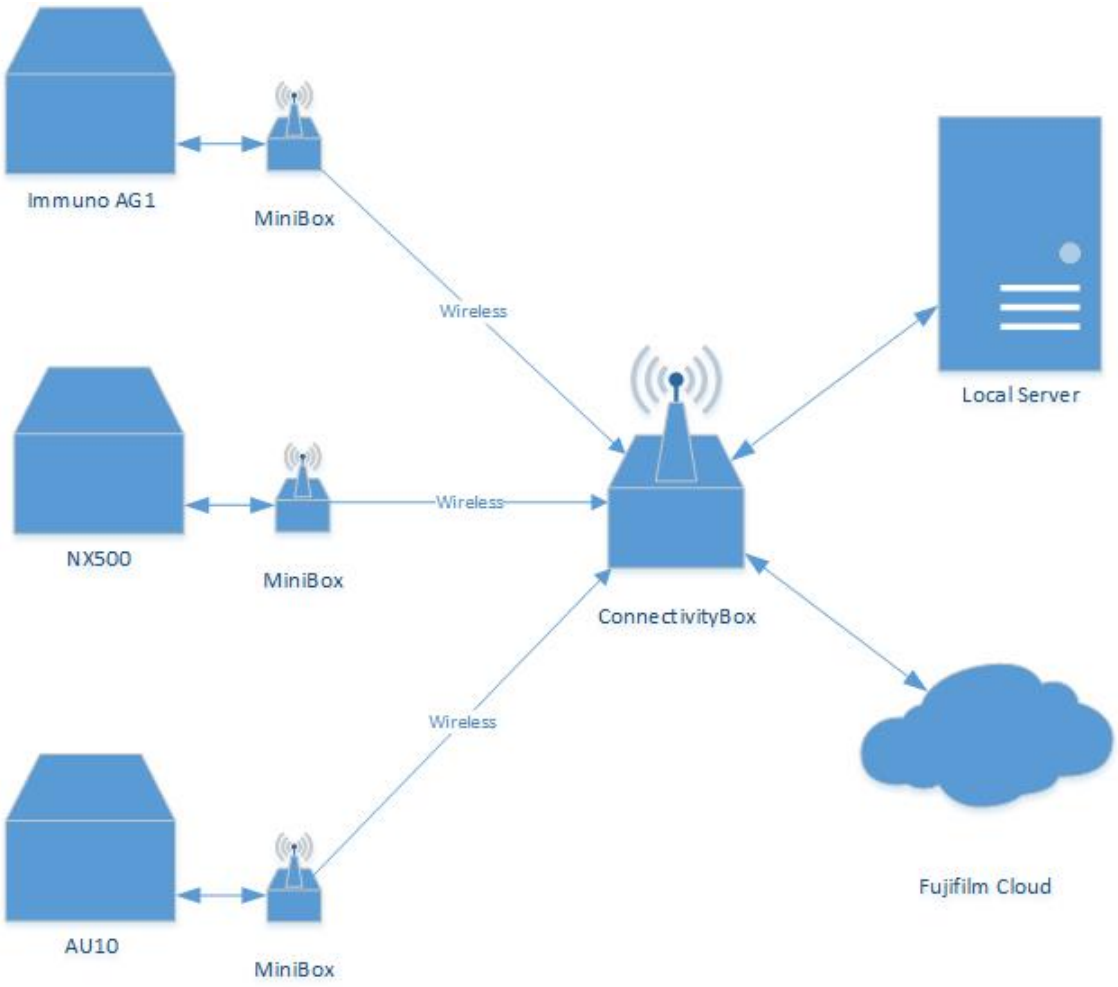

Figura 5-6 – Design Arquitetural – Iteração 3 - *MiniBox*

5.2 Funcionalidades

Após a definição da arquitetura, foi elaborada uma reunião com os representantes dos departamentos de IVD da FUJIFILM para determinar as funcionalidades necessárias para um sistema como este.

Foi-se concluído que existem principalmente três intervenientes no processo de utilização desta *box*:

- Cliente
- Administrador de Sistemas
- Médico

Cada um dos intervenientes têm expectativas diferentes para as funcionalidades das *box*, como por exemplo, o médico quer uma utilização básica da *box* (Figura 5-7), o cliente quer gerir todos os equipamentos em termos de negócio (Figura 5-8) e o administrador de sistema quer ter acesso a manutenção fácil das máquinas IVD (Figura 5-9).

De forma a não repetir funcionalidades nos diagramas, é pressuposto que todas as funcionalidades ficariam disponibilizadas em ambos ambientes locais e ambientes remotos, isto é, na Cloud.

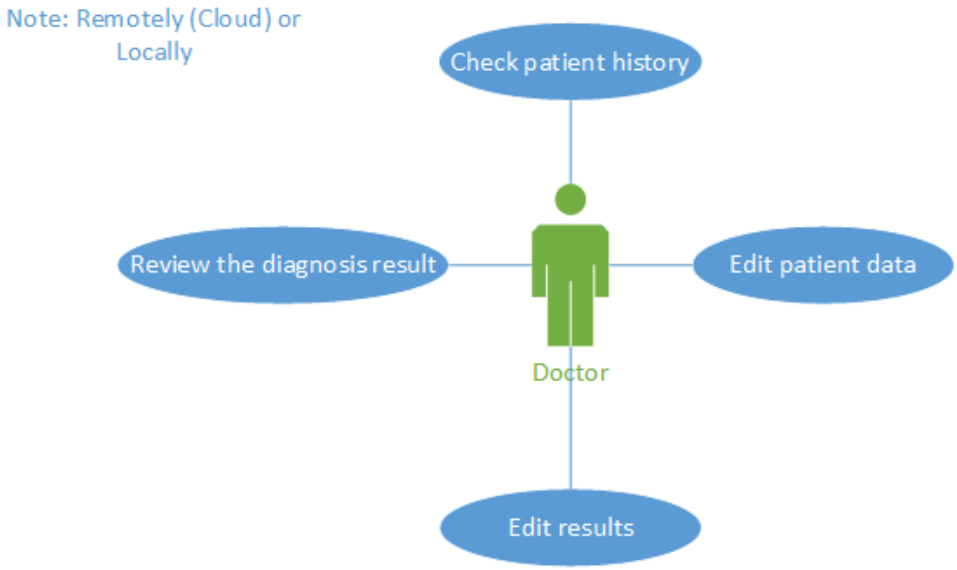

Figura 5-7 – Potencial utilização pela parte do médico

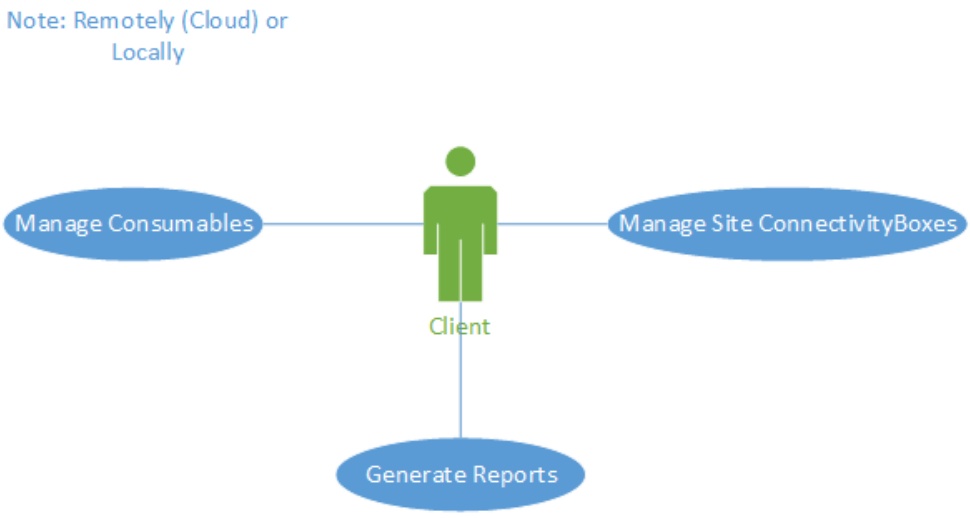

Figura 5-8 – Potencial utilização pela parte do cliente

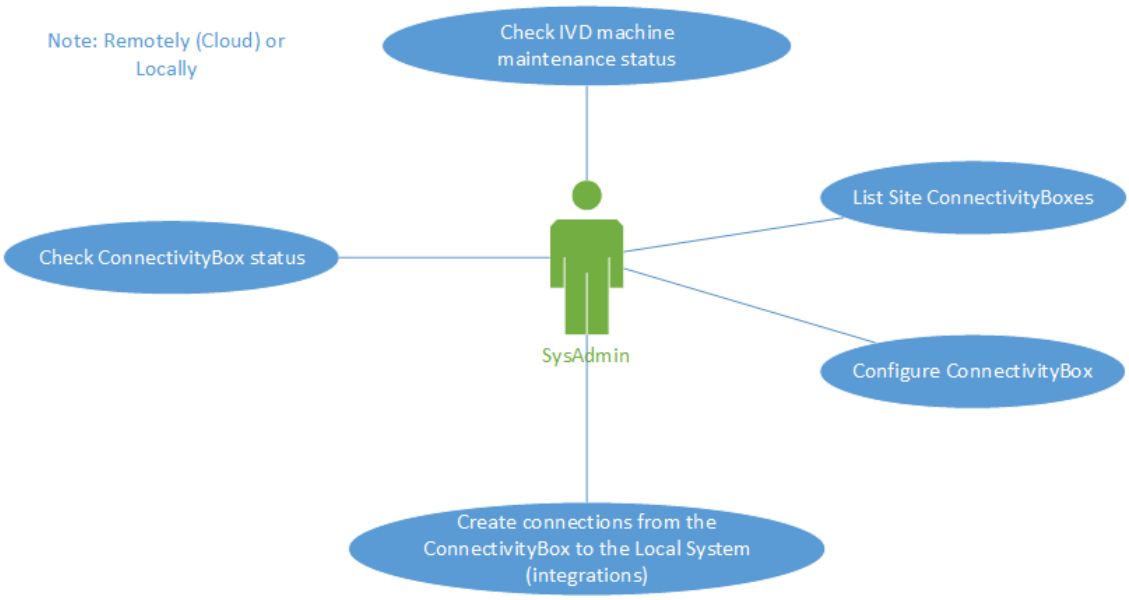

Figura 5-9 - Potencial utilização pela parte do administrador de sistemas

6 Implementação

6.1 Arquitetura Geral

O sistema de conectividade está organizado da seguinte forma (Figura 6-1 – Arquitetura da Solução):

- Uma aplicação a correr na caixa – esta, conforme a sua configuração, liga-se às máquinas IVD, a serviços externos e aos portais de configuração e administração.

- Dois portais web:

  - Um portal de administração de máquinas de conectividade – que contém um sistema de configuração remoto e um sistema de administração geral com monitorização

  - Um portal de visualização de dados gerados pela Connectivity box.

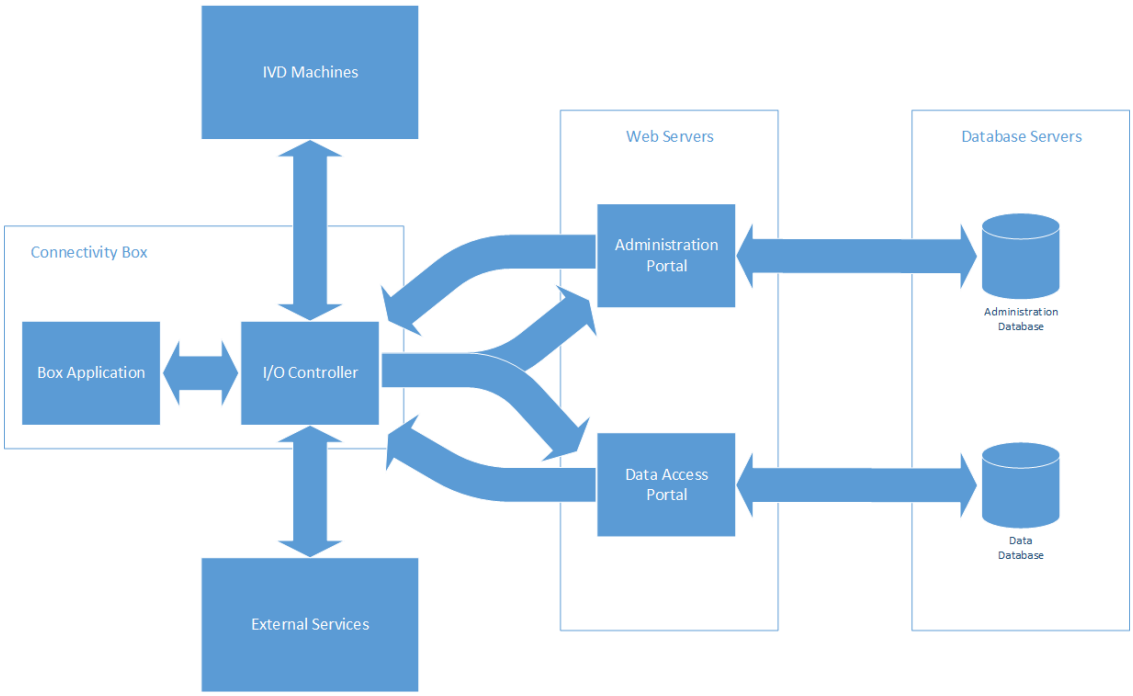

Figura 6-1 – Arquitetura da Solução

6.2 Arquitetura da Aplicação da Box

A aplicação, visto ser o ponto inicial do processo da recolha de dados, necessita de ter as seguintes características:

- Flexível
- Durável/De confiança
- Segura

Para estas características serem alcançadas, a arquitetura inicial da aplicação tem de ser pensada de forma a acomodar tais.

Para tal, foi desenhado um sistema de plugins em que o utilizador escolhe o que necessita e combina os vários plugins de forma a alcançar o efeito desejado. Cada plugin é uma funcionalidade desejada pelo utilizador, como "input via porta serial", "transformação dos dados em HL7" e "envio de dados por email". Por exemplo, a combinação destes três poderia criar um *pipeline* que solucionava o problema do utilizador.

6.3 Detalhes de Hardware

Neste projeto foi necessário também desenhar uma plataforma de *hardware* para servir de base para a aplicação.

Para facilitar o desenvolvimento inicial da aplicação e para acelerar o processo de burocracia associado a compras de bens e serviços, decidiu-se utilizar o SoC da Samsung ARTIK 530 – juntamente com a sua *development board* (Samsung, n.d.).

Para além das razões nomeadas, esta placa foi também escolhida por conter um sistema opensource de fácil modificação. Sendo esta board da Samsung, esta também é facilmente produzida em massa.

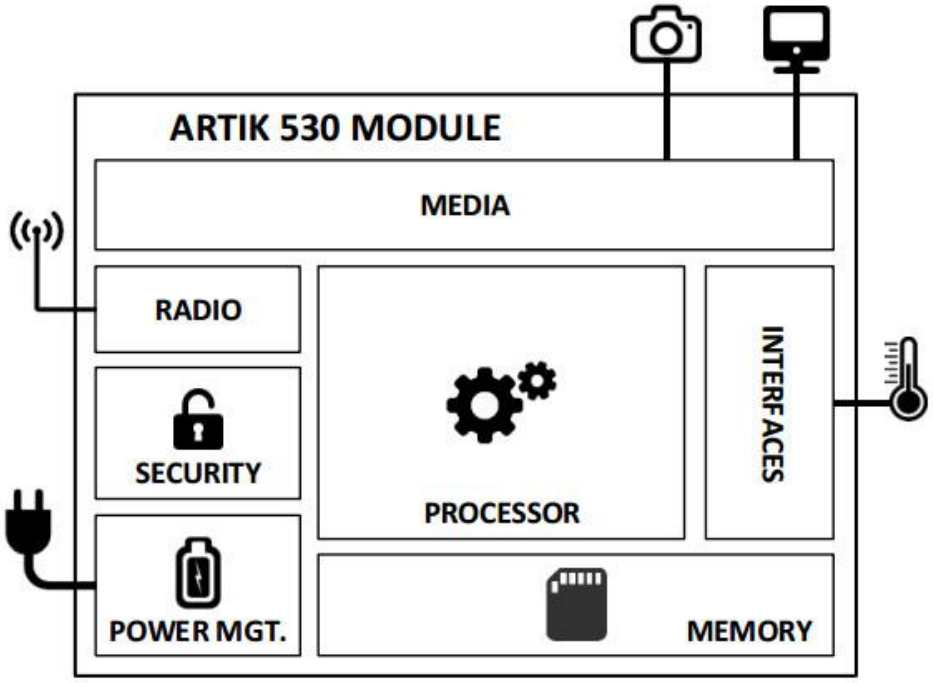

Figura 6-2 – Especificação da *board* ARTIK 530

A placa contém elementos rápidos suficientes para conseguir processar os dados enviados pelas máquinas de IVD, reprocessá-los e transforma-los para o envio para aplicações externas. A Artik 530 contém (Figura 6-2 – Especificação da *board* ARTIK 530) (Samsung, 2016):

- CPU: ARM Cortex A9 – Quad-core a 1.2GHz
- GPU: ARM Mali GPU
- DRAM: 512MB de DDR3
- FLASH: 4GB de eMMC v4.5

Esta placa tem também todos os componentes necessários à criação de placa de expansão de conectividade para o ambiente de produção com interfaces com o exterior (Figura 6-3 – Conexões da *board*) como:

- 1 Ethernet (RJ45)
- 1 WIFI (IEEE 802.1a/b/g/n)
- 1 Serial Port (RS232)
- 1 USB 2.0 (Type A)
- 1 Proprietary connection for Sensors.

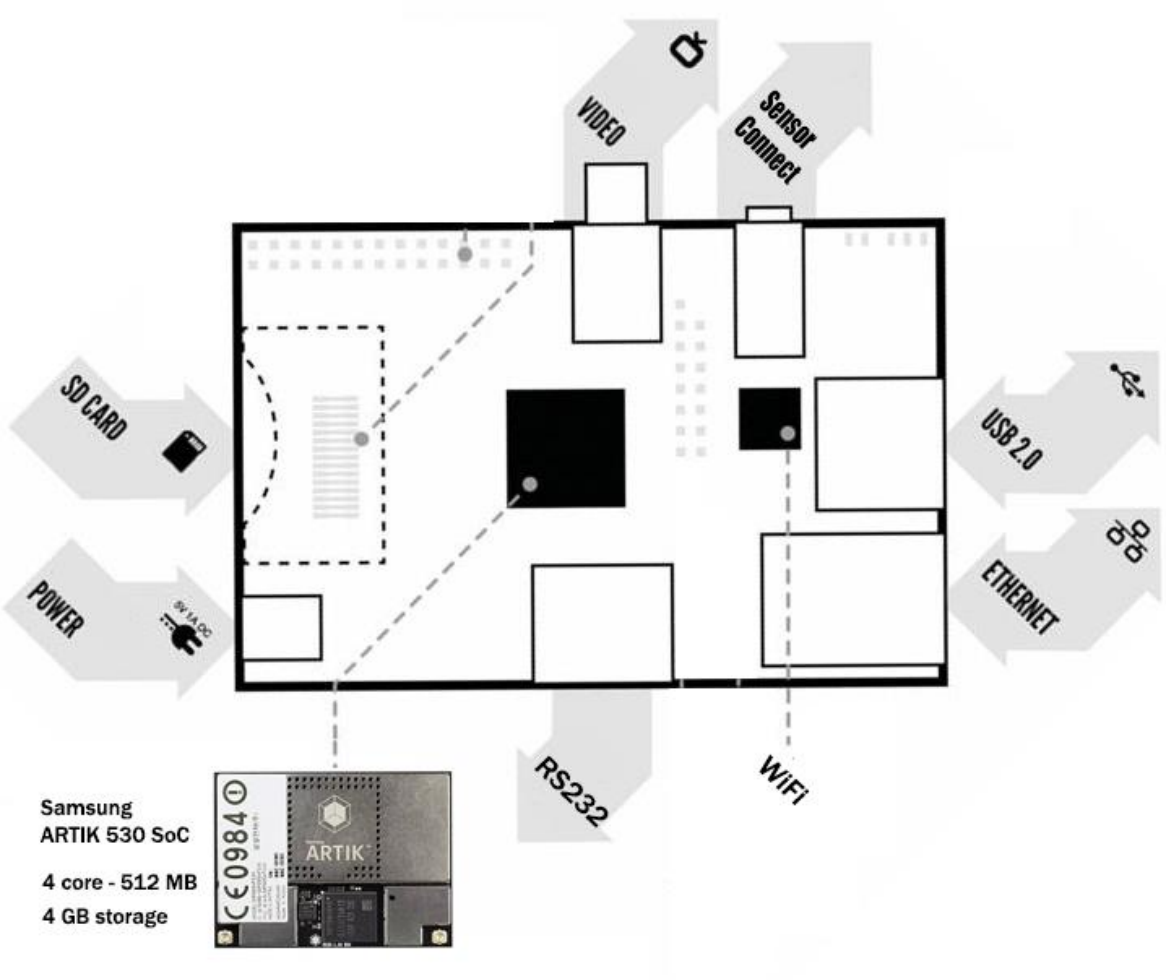

Figura 6-3 – Conexões da *board*

6.4 Restrições Software/Hardware

Para a implementação do sistema que irá correr na ARTIK 530, existem fatores a ter em atenção:

- A aplicação não deverá passar os 256MB de RAM – a ultrapassagem deste limite pode ter impacto negativo sobre a performance do sistema total.
- A aplicação não deverá passar os 512MB de dados persistentes no disco eMMC – apesar de um risco inferior à ultrapassagem de memória RAM, a ultrapassagem deste limite pode levar a um sistema lento e tempos de resposta elevados.
- A aplicação não deverá manter o CPU a 100% durante tempos muito extensos – apesar de não ter um risco elevado, a possibilidade de o sistema sobreaquecer existe, caso não tenha arrefecimento suficiente. A não utilização do CPU a 100% durante muito tempo também deverá aumentar o tempo de vida da placa.

6.5 Connectivity Configurator (Proof of Concept)

Para a elaboração deste produto, foi necessário implementar um sistema de configuração flexível para sustentar todos os tipos de atividade possível dos médicos, no entanto, existe também a necessidade de ser de fácil configuração e necessidade de ser facilmente modificado em qualquer computador ou telemóvel.

Para este fim, optou-se pela utilização de HTML5, visto executar na maioria dos *browsers* modernos – e para a verificação da possibilidade da criação do sistema de configuração, foi feito um protótipo deste, utilizando HTML5 puro.

Decidiu-se também que este configurador é apresentado ao utilizador sob a forma de um género de fluxograma (Figura 6-4 – Protótipo de interface do Connectivity Configurator), à semelhança do *software* de IoT, Node-RED da IBM (Figura 6-5 – Sistema Node-Red da IBM).

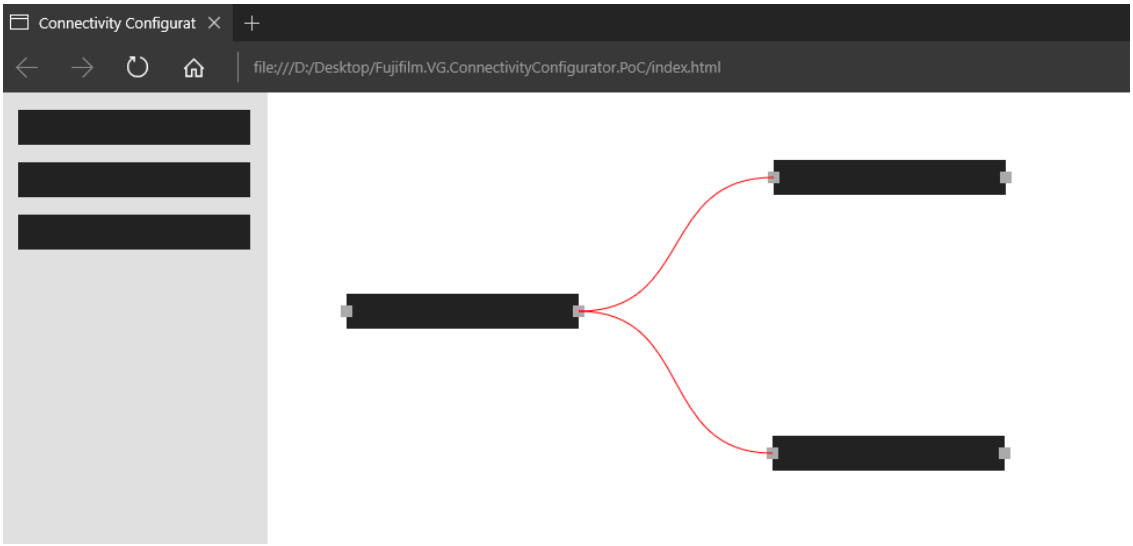

Figura 6-4 – Protótipo de interface do Connectivity Configurator

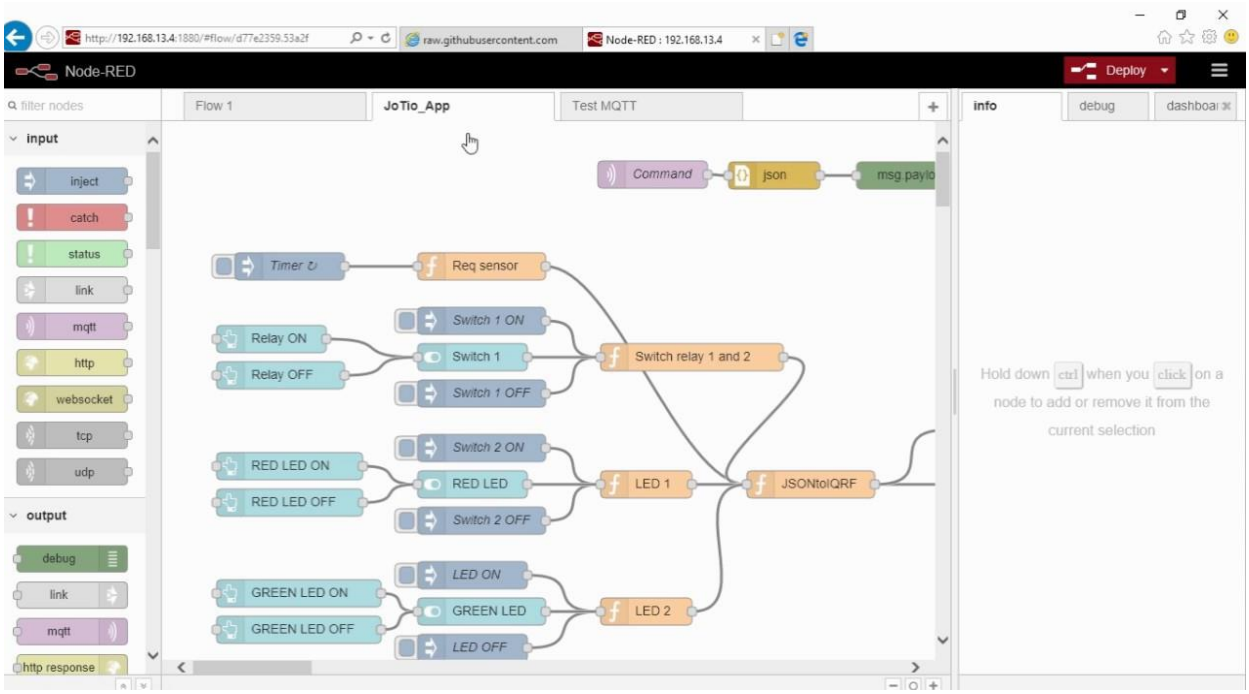

Figura 6-5 – Sistema Node-Red da IBM

Funcionalidades necessárias para validar a utilização de HTML5 no protótipo:

- 2 secções principais do editor:
    - Sidebar com placeholder de plugins
    - Espaço principal para a configuração
- Gerar instâncias de plugins no setor principal do editor (arrastando da sidebar para o setor principal),
- Apagar instâncias de plugins (arrastando do setor principal para a sidebar)
- Gerar conexões entre as instâncias dos plugins,
- Movimentar as instâncias dos plugins sem quebrar ligações,
- Movimentar todas as instâncias dos plugins ao mesmo tempo (*world drag*)

Na implementação deste protótipo utilizou-se uma arquitetura geralmente vista nos motores de jogos single-threaded – o sistema torna-se responsivo seguindo uma seguinte regra: em primeiro lugar, procura modificações no *input* do utilizador, e conforme o resultado desta procura, modifica os dados necessários e fazendo render à cena com estes dados. Estas funções são respetivamente conhecidas por Input, Update e Render (ou Draw) (Figura 6-6 – Funções de Editor).

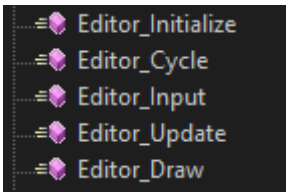

Figura 6-6 – Funções de Editor

Para criar os nós mostrados no editor, usou-se um sistema de entidades. Cada uma das entidades tem associado um tipo de dado utilizado pelo sistema de configuração, como um nó de configuração ou uma ligação entre nós (Connection e ConnectorNode) e estes têm uma ligação ao objeto lógico que descreve o plugin e a instância de plugin.

A função Editor_Initialize é chamada aquando a finalização do load da página HTML5, que por sua vez cria um temporizador para a execução da função Editor_Cycle de milissegundo a milissegundo. Esta função chama então o combo Input, Update e Draw. Esta função, na versão final do editor, também inicializa os plugins a utilizar na sidebar – com possibilidade de adquirir estes dados a partir de uma base de dados remota.

Na inicialização faz-se também o setup dos callbacks de *input* do utilizador – estes geram os dados processados pela função Editor_Input. Os callbacks necessários para este protótipo são as funções de rato e funções de teclado, vistas na Figura 6-7 – Funções de Input.

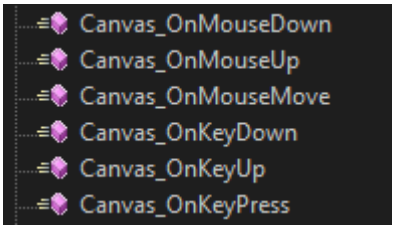

Figura 6-7 – Funções de Input

A função Editor_Input, utilizando os dados modificados pelos callbacks, tem a tarefa de actualizar as entidades, como por exemplo, um nó é movido caso o rato tenha sido clicado, movido e o raycast do rato para o editor intersecta uma das entidades.

Seguindo para a função Editor_Update, neste protótipo, de forma a demonstrar a capacidade da criação de um editor simples, apenas se faz o *sort* das *layers* (

```
function Editor_Update(editor)
{
    Canvas_Update(editor.canvas);

    // Sort Entities
    {
        editor.entities.sort(function (a, b)
        {
            var valueA = 0;
            var valueB = 0;

            if(a.transparent)
            {
                valueA += 100;
            }
            if (b.transparent)
            {
                valueB += 100;
            }
            if(a.volatile)
            {
                valueA += 200;
            }
            if (b.volatile)
            {
                valueB += 200;
            }
            valueA += a.position.z;
            valueB += b.position.z;

            if (valueA == valueB)
            {
                valueA += a.id;
                valueB += b.id;
            }

            return valueA - valueB;
        });
    }
}
```

Figura 6-8 – Função de organização de *Layers*) e entidades de forma a serem desenhadas de forma correta sem artefactos de "z-buffer" (visto o editor usar um render de software, não é

um z-buffer propriamente, mas é comparável a tal). Este *sort* utiliza as propriedades dos objetos Entity, *transparent*, volatile (true se a entidade for temporária – como a ligação de um nó para outro nó aquando a sua criação), a posição no eixo Z e o ID da entidade (para criar uma solução constante no caso da soma de todas as outras propriedades ser equivalente) para ordenar o render de trás para a frente sendo as entidades mais importantes as ultimas a serem desenhadas.

Figura 6-8 – Função de organização de *Layers*

A função de desenho, Editor_Draw, apenas itera por todas as entidades e desenha-as: quadrados (Canvas_Draw_Rect) para as instâncias de plugins, pontos (Canvas_Draw_Point) para os nós de início e fim de conexão, curvas de bezier (Canvas_Draw_Bezier) or quadráticas (Canvas_Draw_Quadratic) para os conectores de plugins, funções estas vistas na Figura 6-9 – Funções de Desenho no Canvas.

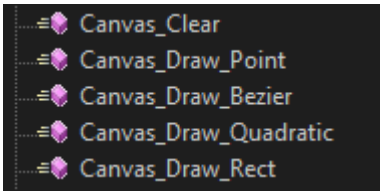

Figura 6-9 – Funções de Desenho no Canvas

Toda a aplicação pode ser descrita em termos de organização de dados pelo diagrama de classes na Figura 6-10 – Diagrama de Classes do protótipo do configurador.

Na versão final do editor, existe uma função que extrai dos dados da cena atual e transforma-os num ficheiro de configuração legível pelo programa principal da caixa de conectividade.

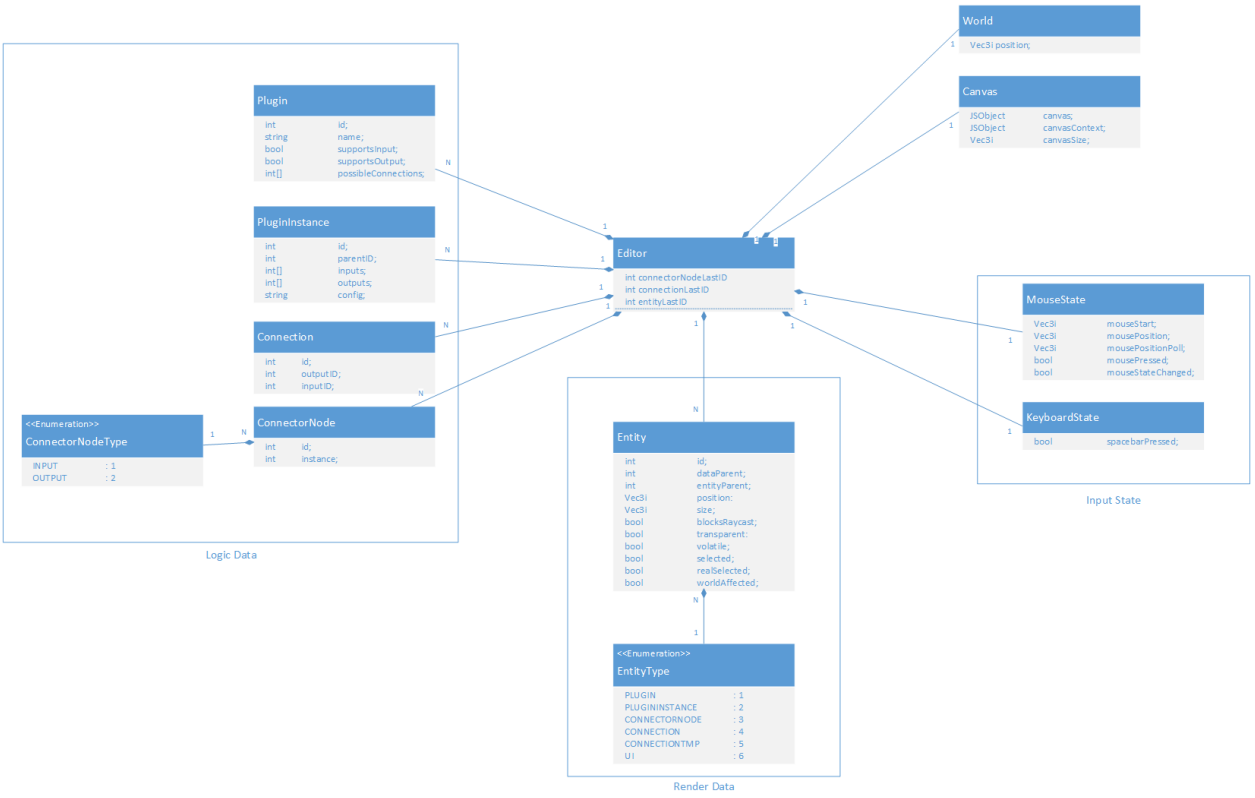

Figura 6-10 – Diagrama de Classes do protótipo do configurador

6.6 Connectivity Box

6.6.1 Implementação geral

Para o programa que corre na box de conectividade foram ponderadas várias possibilidades em termos de linguagem de programação principal:

- Python
- C# (utilizando .NET Core)
- Java
- C
- C++

As linguagens Python, C# e Java foram avaliadas abaixo das expetativas e descartadas imediatamente pelas seguintes razões:

- C#, utilizando .NET Core numa plataforma ARM, é de difícil manutenção, pois na altura de escrita, a base .NET Core é modificada constantemente com alterações significativas.
- C# e Java são de fácil descompilação e Python tem o código aberto sem qualquer tipo de compilação prévia (ver Figura 6-11 – Código de um programa de *download* de e Figura 6-12 – Código de um programa de *download* de vídeos (adquirido por descompilação) – a primeira imagem consistindo no código real de uma aplicação em C# e a segunda consiste no código descompilado dessa mesma aplicação).
- O código em C#, Java e Python pode ser ofuscado no processo de *release*, no entanto, mesmo um atacante com conhecimentos básicos consegue dar *reverse engineer*.
- Mantendo a estrutura de plugins, todas as três linguagens não oferecem segurança suficiente à modificação de plugins, mesmo estes estando assinados com strong-keys (no caso de C#). (Ver Figura 6-13 – Código de uma DLL de teste, Figura 6-14 – Código descompilado da DLL de teste e Figura 6-15 – Código modificado da DLL de teste – sendo a imagem 6-13 o código não modificado de um DLL assinado com uma strong-key, a imagem 6-14 o código descompilado e o respetivo código MSIL e a imagem 6-15 o código do DLL modificado de forma a retornar sempre true – funcional mesmo com assinatura de DLL).

Em contraste, C e C++ são:

- Estáveis, mesmo numa plataforma ARM,
- A descompilação de binários de C e C++ é complicada, pois é necessário um conhecimento extenso na plataforma em que o binário esta a correr e o tipo de código assembly gerado pelo descompilador (ver Figura 6-16 – Programa em C++ descompilado).
- Não é necessário obfuscar o código pois este é um passo inerente ao processo de compilação de binários.
- A modificação de DLLs ainda é possível, no entanto, esta tarefa é exponencialmente mais complexa devido à complexidade do entender o programa a partir de código descompilado.

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using YoutubeExtractor;

namespace YoutubeExtractorV2
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] yout = File.ReadAllLines("input.txt");

            Directory.CreateDirectory("Songs");

            foreach (var item in yout)
            {
                try
                {
                    Console.WriteLine(item);

                    IEnumerable<VideoInfo> videoInfos = DownloadUrlResolver.GetDownloadUrls(item);

                    if (videoInfos.Count() == 0) continue;

                    VideoInfo video = videoInfos
                        .Where(info => info.Resolution == 480 && info.VideoExtension == ".mp4")
                        .FirstOrDefault();

                    if (video == null) continue;

                    if (video.RequiresDecryption)
                    {
                        DownloadUrlResolver.DecryptDownloadUrl(video);
                    }

                    var videoDownloader = new VideoDownloader(video, Path.Combine("Songs", video.Title + video.VideoExtension));

                    videoDownloader.DownloadProgressChanged += (sender, args1) => Console.WriteLine(args1.ProgressPercentage);

                    videoDownloader.Execute();
                }
                catch (Exception) { }
            }


        }
    }
}
```

Figura 6-11 – Código de um programa de *download* de vídeos

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using YoutubeExtractor;

namespace YoutubeExtractorV2
{
  internal class Program
  {
    private static void Main(string[] args)
    {
      string[] strArray = File.ReadAllLines("input.txt");
      Directory.CreateDirectory("Songs");
      foreach (string videoUrl in strArray)
      {
        try
        {
          Console.WriteLine(videoUrl);
          IEnumerable<VideoInfo> downloadUrls = DownloadUrlResolver.GetDownloadUrls(videoUrl, true);
          if (downloadUrls.Count<VideoInfo>() != 0)
          {
            VideoInfo videoInfo = downloadUrls.Where<VideoInfo>((Func<VideoInfo, bool>) (info =>
            {
              if (info.Resolution == 480)
                return info.VideoExtension == ".mp4";
              return false;
            })).FirstOrDefault<VideoInfo>();
            if (videoInfo != null)
            {
              if (videoInfo.RequiresDecryption)
                DownloadUrlResolver.DecryptDownloadUrl(videoInfo);
              VideoDownloader videoDownloader = new VideoDownloader(videoInfo, Path.Combine("Songs", videoInfo.Title + videoInfo.VideoExtension), new int?());
              videoDownloader.DownloadProgressChanged += (EventHandler<ProgressEventArgs>) ((sender, args1) => Console.WriteLine(args1.ProgressPercentage));
              videoDownloader.Execute();
            }
          }
        }
        catch (Exception ex)
        {
        }
      }
    }
  }
}
```

Figura 6-12 – Código de um programa de *download* de vídeos (adquirido por descompilação)

```
namespace HackTest_02
{
    public class Class1
    {

        private static string Password = "Isto é um test";

        public static bool Test(string data)
        {

            if (data.Equals(Password))
            {
                return true;
            }

            return false;

        }

    }
}
```

Figura 6-13 – Código de uma DLL de teste

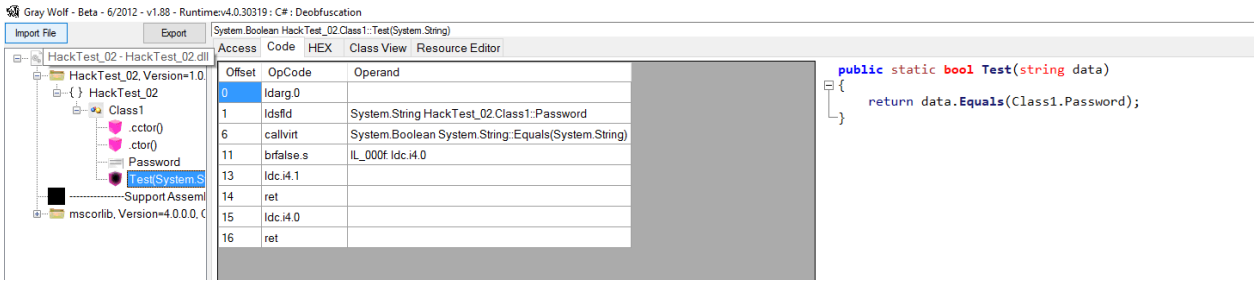

Figura 6-14 – Código descompilado da DLL de teste

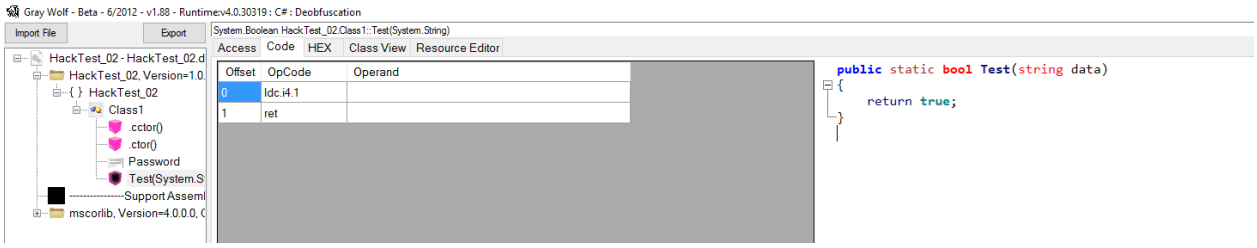

Figura 6-15 – Código modificado da DLL de teste

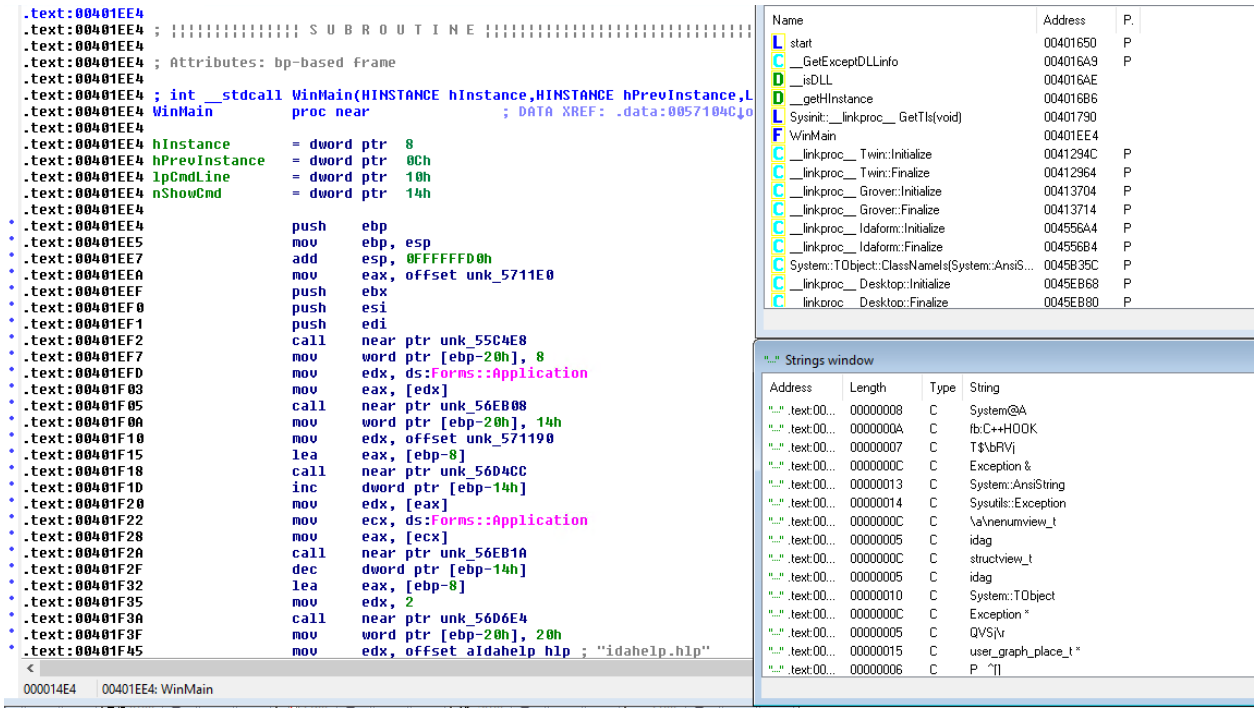

Figura 6-16 – Programa em C++ descompilado

6.6.2 Arquitetura

Em termos da arquitetura do serviço principal a correr na Box, decidiu-se, como referido anteriormente, utilizar um sistema de plugins.

Este sistema de plugins, conforme mostrado na Figura 6-17 – Arquitetura do sistema de plugins, trás inúmeras vantagens em relação a sistemas mais tradicionais, sendo estas:

- Flexibilidade – Funcionalidades dinâmicas de acordo com a necessidade do utilizador
- Segurança – Com a flexibilidade desse sistema, não existe a necessidade de todos os plugins estarem localizados na box – estes podem ser transferidos conforme a necessidade do utilizador. Isto cria uma superfície de ataque inferior à criada por o acesso constante a todo o código.
- Minimização da *footprint* – Visto a capacidade de armazenamento da Box ser pequena, a não disponibilização de todo o código constantemente minimiza o espaço ocupado por binários.
- *Upgradability* – Facilidade de troca de código antigo por uma versão mais recente.
- *Debugability* – Devido ao código estar separado modularmente, o *debug* de erros durante a implementação de funcionalidades torna-se mais fácil.

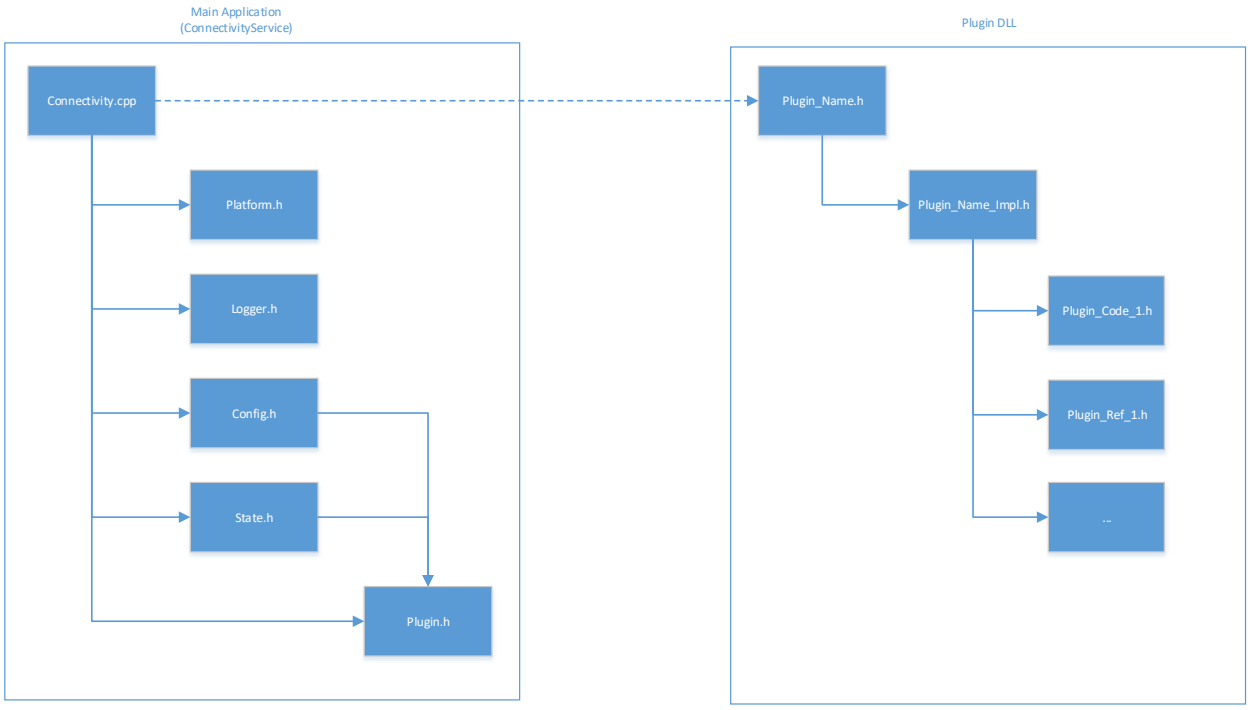

Figura 6-17 – Arquitetura do sistema de plugins

Para manter a coerência no acesso aos plugins, foi decidida uma interface de implementação necessária pela parte dos plugins (Figura 6-18 – Interface de entrada num plugin). Esta interface trás comunicação bidirecional entre os plugins e a aplicação principal.

```
void * Load            (char * config, uint32 length);
void * Start           (void * state);
void * Process         (void * state, void * data);
void * Dispose         (void * state, void * data);
bool   Stop            (void * state);

int32  StateCode       (void * state);
char * StateOverview   (void * state);
int32  ErrorCode       (void * state);
char * ErrorOverview   (void * state);
```

Figura 6-18 – Interface de entrada num plugin

Esta interface está localizada normalmente no par de ficheiros [Plugin_Name.h] e [Plugin_Name.cpp] sendo estes o ponto de entrada do plugin.

Explicando brevemente a funcionalidade relacionada com esta interface:

- O Load é a função inicial do plugin. Esta gera os dados necessários para o funcionamento do plugin, retornando-os sobre a forma de uma zona de memória anónima.
- Start é a função que inicia todas as tarefas assíncronas necessárias ao plugin – como por exemplo, tarefas de coleção de dados, gestão de memória interna,...
- Process é a função principal do plugin – esta função recebe o estado atual do plugin e os dados prontos a serem processados por este plugin – retorna os dados processados.
- A função Dispose é a função que necessita de ser chamada após a passagem dos dados retornados pela função Process a outras entidades do sistema. Esta função, caso a função Process aloque memória, irá limpa-la e caso necessário dealoca-la.
- Stop pára e limpa os dados do plugin.
- Os métodos StateCode/StateOverview/ErrorCode/ErrorOverview são métodos mais utilizados para introspeção do programa, para este saber se está a correr sem falhas, ou se estiver, tentar recuperar a situação.

Estes plugins, para criar uma organização categórica das suas funcionalidades, foram divididos nas seguintes categorias:

- Input – tipo de plugin especializado em inicializar dados, isto e, recebendo-os do exterior ou da própria box,
- Pré-Processamento – tipo de plugin genérico executado entre o plugin de input e processamento,
- Processamento – tipo de plugin com a logica de tradução dos dados de entrada para os dados de saída,
- Pós-Processamento – tipo de plugin genérico executado entre o plugin de processamento e de output,
- Output – tipo de plugin especializado em guardar dados, isto e, inserindo-os num sistema externo.

A união dos 3 principais plugins – Input, Processamento e Output – cria o *pipeline* de dados (Figura 6-19 – União de plugins para criação de um *pipeline*) do sistema, tendo a possibilidade de ser estendido com os 2 outros tipos de plugin – Pré-processamento e Pós-processamento.

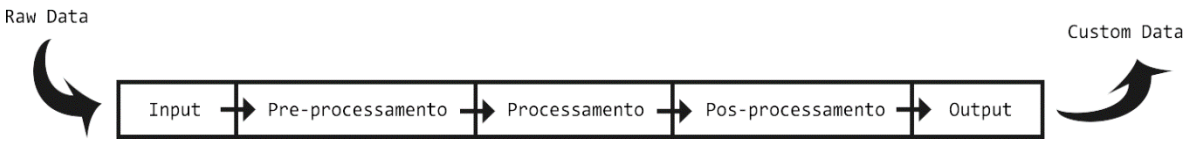

Figura 6-19 – União de plugins para criação de um *pipeline*

6.6.3 Serviço base

O serviço base a correr na box contém a logica principal de como é que os plugins são interligados de forma a criarem um pipeline de transformação de dados. Este tem como tarefa:

- A receção da configuração criada pelo utilizador (feita na pagina web de configuração),
- O download dos plugins correspondentes à configuração,
- A inicialização correta dos plugins,
- Manutenção das linhas de comunicação entre os plugins.

6.6.3.1 Funcionamento

O funcionamento deste serviço tem por base a lógica encontrada na Tabela 6-1 – Pseudo-código da inicialização do sistema base.

Tabela 6-1 – Pseudo-código da inicialização do sistema base

```
Início

      Download da última versão da configuração

      Para cada um dos plugins na configuração
            Se o plugin não existe no filesystem
                  Download do plugin
            Fim se
      Fim para

      Para cada um dos plugins na configuração
            Se o plugin não estiver ligado dinamicamente
                  Ligar o .SO do plugin
            Fim se
      Fim para

      Para cada um dos plugins na configuração
            Chamar plugin.Load()
            Chamar plugin.Start()
      Fim para

      Para cada uma das linhas de comunicação entre plugins na
configuração
            Spawn thread com a logica de comunicação de plugins
      Fim para
```

```
      Esperar pelo término de todas as linhas de comunicação

Fim
```

O funcionamento de uma linha de comunicação também tem por base uma lógica simples, mostrada na Tabela 6-2 – Pseudo-código da comunicação com plugins.

Tabela 6-2 – Pseudo-código da comunicação com plugins

```
Início

      Gerar uma lista dinamica vazia
      Por o plugin com categoria de input na lista dinamica


      Enquanto a lista não estiver vazia
            Chamar plugin.Process() no primeiro plugin da lista com
os dados gerados pelo plugin anterior (null no input)
            Adicionar todos os plugins relacionados com o plugin
inicial à lista
            Remover o primeiro plugin da lista
      Fim enquanto

Fim
```

6.6.4 Plugins

Apesar do projeto ser um trabalho em constante evolução, foi decidida a criação de um set inicial de plugins disponibilizados, com base nas 3 classes principais de plugins. Estes plugins são:

- Input / Output
    - Injector
    - Logger
    - MLLP Client
    - MLLP Server
    - ODBC
    - Serial Port
    - WebAPI
    - Monitor

- Processadores
    - AG1
    - NX500
    - HL7
    - Mapper

6.6.4.1 Injector

O plugin Injector é o plugin mais básico em termos de funcionalidade. Este plugin, categorizado como Input apenas, envia dados escolhidos pelo utilizador para os plugins seguintes.

Em termos de implementação, o Injector funciona mantendo o timestamp da última chamada à função Process e determina quantas mensagens deve enviar para plugin seguinte. O timestamp é então atualizado e o sistema pode prosseguir com o *pipeline*.

Este sistema não utiliza threads nem bibliotecas externas.

6.6.4.2 Logger

Como o nome indica, o plugin Logger é utilizado para fazer log da dados intermédios no *pipeline*, isto é, este plugin pode ser caracterizado como Output, mas pode ser aplicado na categoria de pre-processamento e pos-processamento.

Este plugin é para uso interno apenas – não existindo a necessidade de haver uma otimização agressiva.

Tal como o plugin Injector este também tem presente uma implementação simplista, em que este conforme as configurações iniciais, os dados recebido pela função Process são escritos para disco, para consola e/ou enviados por rede num pacote UDP.

Seria possível criar um thread para não bloquear a função Process utilizando queues de comandos, mas decidiu-se esta implementação para manter a simplicidade.

6.6.4.3 MLLP Client

MLLP Client é um plugin de Output de dados utilizando o *standard* HL7, mais especificamente o standard de comunicação MLLP.

Tal como os dois anteriores plugins, este também tem a sua principal logica embebida na função Process.

Conforme a configuração, este plugin pode-se tentar ligar ao servidor MLLP apenas uma vez, ou ligar-se e desligar-se sempre que existe a necessidade de comunicação. Esta ligação é estabelecida utilizando sockets de berkley.

Caso a ligação não consiga ser estabelecida ao servidor passado 3 tentativas, o sistema irá armazenar a informação recebida e activar um sub-sistema de aviso ao administrador de

sistemas. Caso a ligação ser estabelecida corretamente, os dados recebidos como parâmetros da função Process são enviados para o servidor (Figura 6-20 – Fluxograma da lógica do plugin MLLP Client).

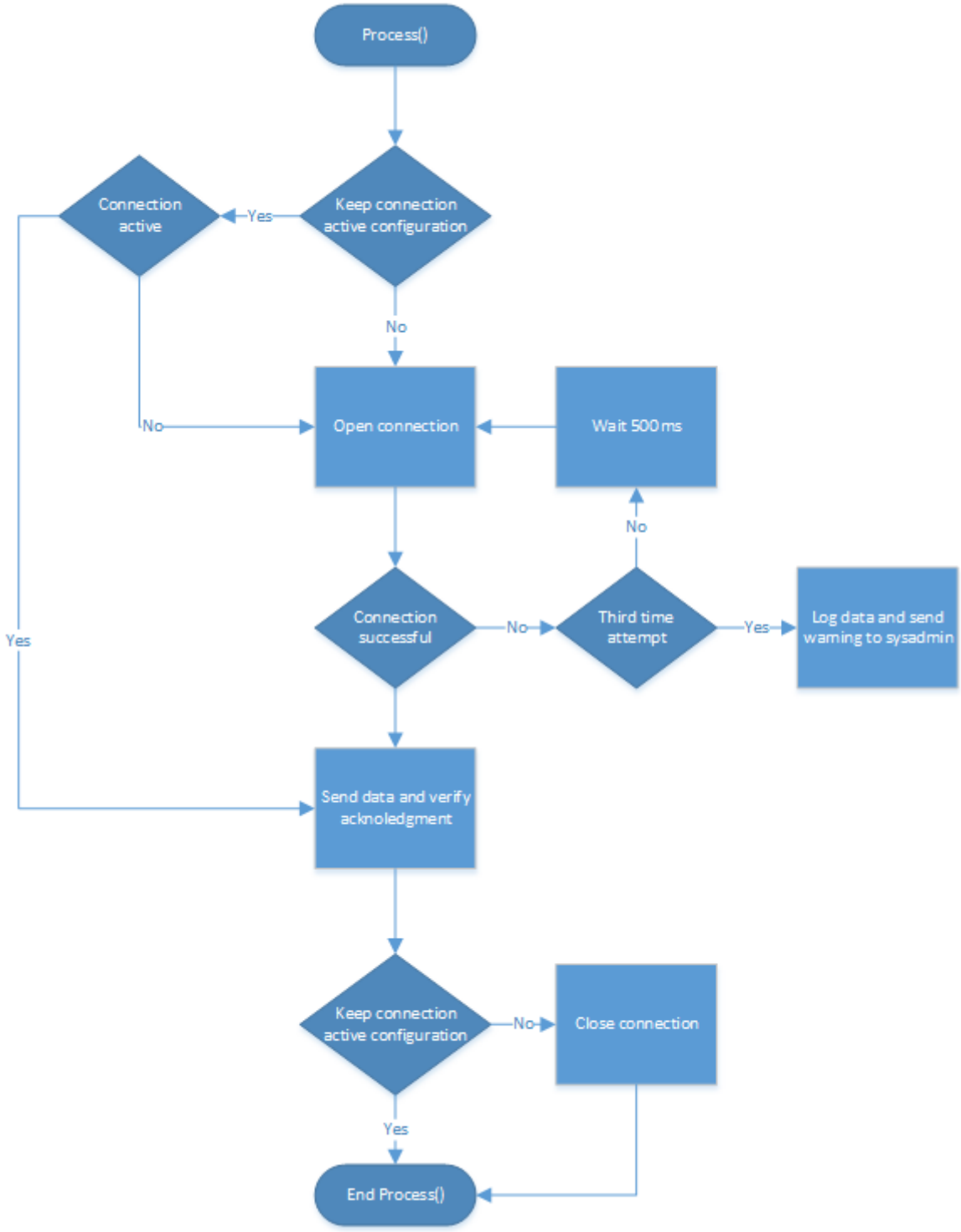

Figura 6-20 – Fluxograma da lógica do plugin MLLP Client

6.6.4.4 MLLP Server

O plugin MLLP Server funciona de forma semelhante ao plugin MLLP Client, no entanto este é um plugin de Input.

O plugin mantém uma porta aberta à escuta de comunicações num thread a parte que recebe os dados e os armazena num buffer local. Este buffer é acedido por dois threads, o da comunicação e o thread principal do pipeline. Esta interação é explicada no fluxograma Figura 6-21 – Fluxograma da lógica do plugin MLLP Server.

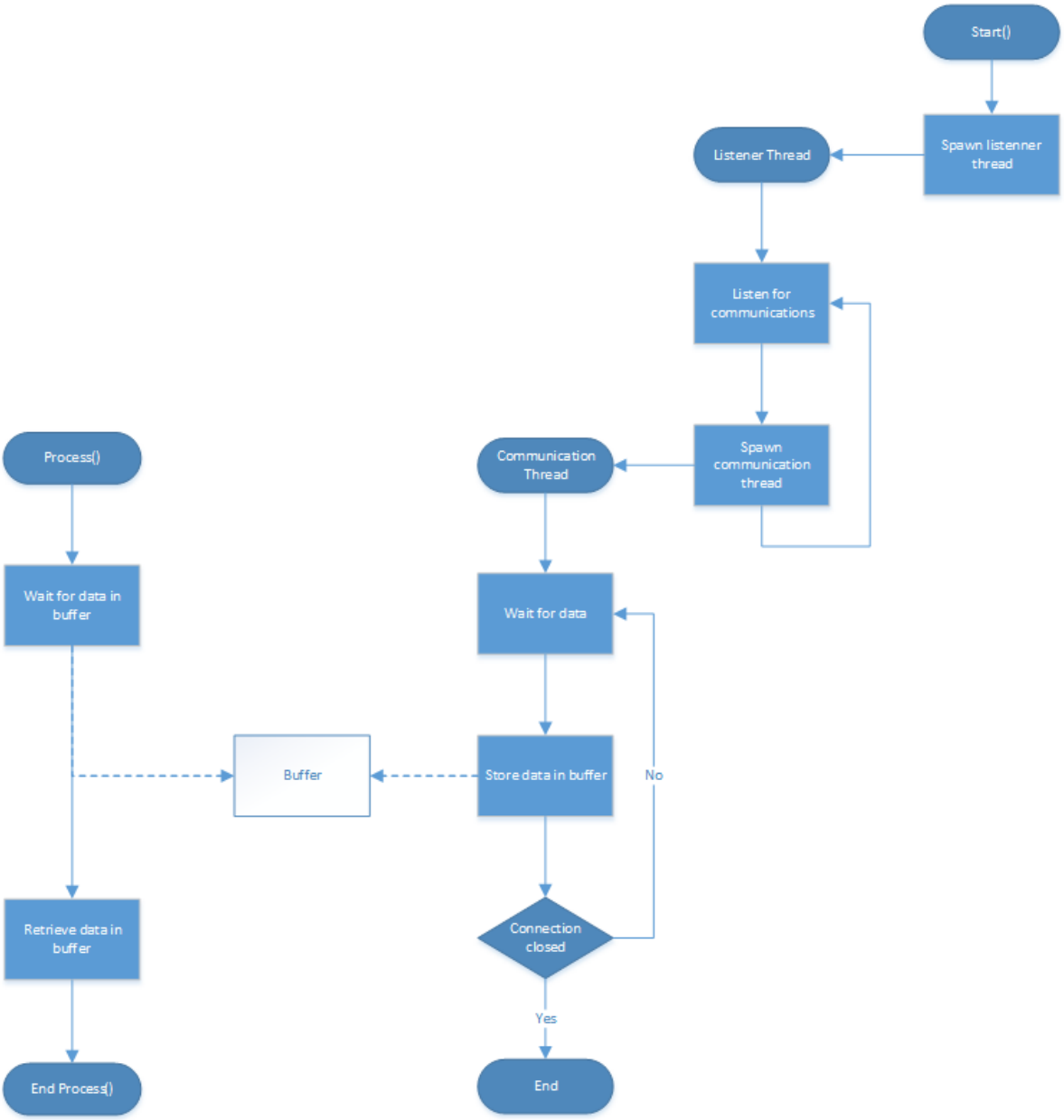

Figura 6-21 – Fluxograma da lógica do plugin MLLP Server

6.6.4.5 ODBC

Este plugin adiciona a possibilidade de enviar dados para uma base de dados externa de forma organizada – um plugin de output.

Nesta primeira versão, este apenas contém logica para a inserção direta de uma linha numa tabela, sem possibilidade de concatenação de dados com outros dados já existentes na base de dados ou ligação a dados também já existentes.

Em termos de implementação este plugin tem duas partes principais, a inicialização e o processo normal.

A inicialização tem a principal função de ativar os componentes ODBC do sistema Linux da board e verificar – utilizando um comando SQL "Select" – se a tabela identificada pelo utilizador na configuração existe.

Já o processamento tem a função de identificar nos dados que foram recebidos, a sua chave que relaciona com uma coluna da tabela. Isto é determinado utilizando mapeamentos do utilizador na configuração, como por exemplo:

```
Dado_1_entrada = Tabela1.Coluna1;
Dado_2_entrada = Tabela1.Coluna2;
Dado_3_entrada = Tabela7.XPTO;
```

A partir deste mapeamento, um comando "Insert" é gerado e executado – no caso de falha, este é posto num modo de "tentativa" em que de 500 em 500ms tenta ser inserido, até ao máximo de 3 tentativas que, se este número for passado, o administrador de sistemas é avisado.

6.6.4.6 Serial Port

O plugin Serial Port é o plugin de ligação à maioria das máquinas IVD atuais. Este lê dados da porta serial da Box e envia-os para o plugin seguinte.

Este funciona da mesma forma do plugin MLLP Server, na forma em que usa um thread para não bloquear a ação do programa principal – no entanto, como só existe uma porta serial na box, apenas um thread é necessário e não um thread por cliente.

Aquando a chamada da função process, esta bloqueia caso não existam dados (até existirem). Os dados recebidos são então retirados do buffer e enviados para o plugin seguinte do pipeline.

6.6.4.7 WebAPI

O plugin WebAPI é um plugin de output de dados para serviços REST externos.

Durante a implementação deste plugin existiu a possibilidade de usar uma biblioteca externa – cURL – ou usar a biblioteca interna do SoC ARTIK.

Decidiu-se utilizar a biblioteca interna devido aos seguintes fatores:

- cURL consegue chegar a ocupar 200 mb em disco.

- a biblioteca interna funciona sem a ocupação de memória RAM, isto é, utiliza uma memória interna.

Em termos de logica, este plugin funciona da mesma forma que o plugin de MLLP Client, em que todo o processo é feito durante a função Process. Conforme a configuração do utilizador, o sistema gera um pedido GET ou POST, utilizando os dados recebidos pela função.

6.6.4.8 Monitor

O plugin monitor é o sistema principal de introspeção da box. Este monitoriza as temperaturas dos componentes da placa, utilização de CPU, disco e memória RAM. Este sistema também armazena dados de uptime e dados de rede como IP e hostname, dados estes que facilitam o trabalho de um administrador de sistemas.

Este plugin é pelicular na forma em que dificilmente consegue ser categorizado nas categorias de input e processamento, pois este é a junção de ambos – recebe dados da placa e processa-os. Pode-se verificar que a utilização deste é principalmente acoplada a um plugin de Log ou WebAPI para a transferência dos dados gerados por este plugin.

6.6.4.9 AG1 / NX500

Os plugins AG1 e NX500 são plugins que processam os dados recebidos por um plugin de Input e retornam esses dados de forma organizada de acordo com o protocolo proprietário de uma destas máquinas.

Este plugin é considerado um plugin de categoria de processamento.

6.6.4.10 HL7 / Mapper

Ambos os plugins HL7 e Mapper funcionam de forma semelhante. Estes têm como função na utilização de dados já processados e reorganiza-los. O plugin HL7 é um plugin extrapolado do plugin Mapper – isto é, funciona da mesma forma, no entanto tem mais restrição em termos de posicionamento de dados.

Os plugins têm a logica principal na função process em que esta, aquando a chamada com parâmetros corretos e verificáveis, extrai os dados destes e insere-os num template fornecido pelo utilizador, como demonstrado na tabela XXX:

```
Template:

MSH|^~\&|${NOME_LABORATORIO}|ELAB-3|GHH
OE|BLDG4||^~\&|${N_PACIENTE}||${TIPO_MENSAGEM}|${TIPO MENSAGEM}|P|2.4

Input:

NOME_LABORATORIO = ANALISE.D.JOAO
N_PACIENTE = MATIAS2912
TIPO_MENSAGEM = ORM^ORM
DADO_NAO_USADO = DADOSTESTE

Output:

MSH|^~\&|ANALISE.D.JOAO|ELAB-3|GHH OE|BLDG4||^~\&| MATIAS2912|| ORM^ORM | ORM^ORM
|P|2.4
```

6.7 Exemplos de *pipeline*

De acordo com as necessidades dos médicos, consegue-se prever a utilização principal dos seguintes *pipelines*:

- Pipeline 1: Serial Port -> AG1 -> HL7 -> MLLP Client

- Pipeline 2: Serial Port -> NX500 -> HL7 -> MLLP Client

Estes *pipelines* transformam os dados recebidos pelas respetivas máquinas IVD em dados de fácil entendimento pela parte de um sistema externo, um RIS ou HIS, utilizando HL7.

Conforme as configurações, o sistema funciona da forma demonstrada pelo fluxograma na Figura 6-22 – Lógica de pipeline completa.

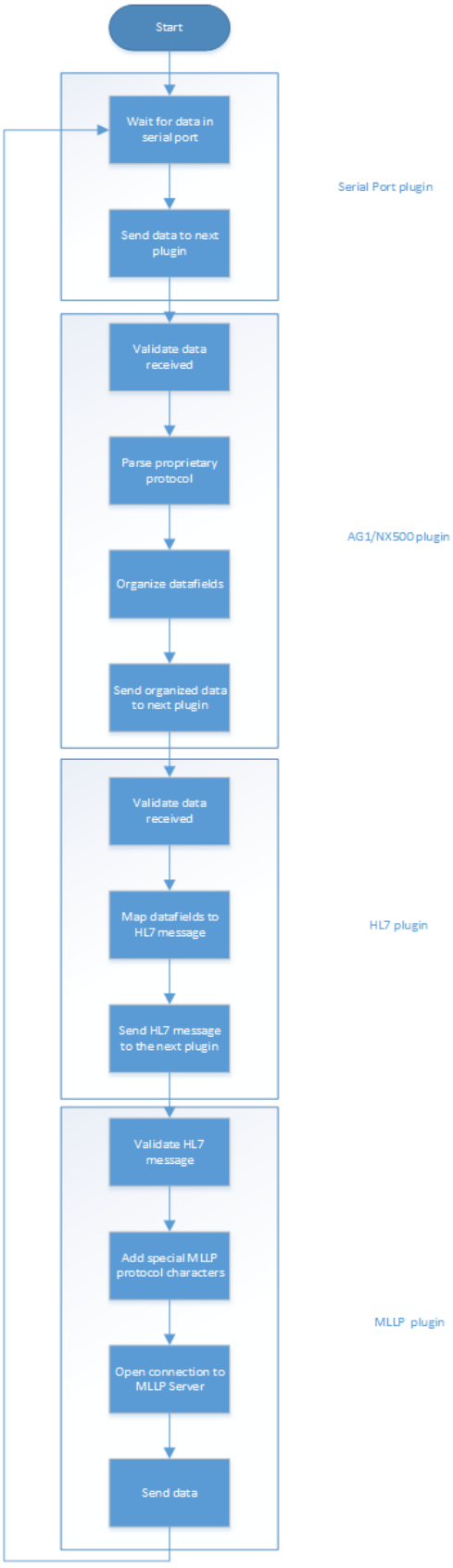

Figura 6-22 – Lógica de pipeline completa

7 Avaliação

7.1 Avaliação Teórica do Produto

O produto será avaliado conforme descrito no capítulo 3.1 Avaliação do Produto.

Este vai ser avaliado na sua estabilidade, pois um caso de falha pode levar a dados errados que podem posteriormente levar a um diagnóstico errado pela parte de um médico.

Para tal avaliação será criado um emulador de tramas que irá alimentar a solução com um número significativo de exemplos, para comparar o nível de precisão da máquina. Também será posto este emulador a enviar dados durante um tempo extenso, de forma a verificar a existência de possíveis mal funcionamentos do sistema. Ambos os níveis de precisão e o tempo de falha terão de ficar abaixo de um certo nível ainda a definir.

Em termos de rapidez, o sistema será utilizado por médicos que depois irão dar feedback após um período de uso a ser determinado. Serão recolhidos os dados de satisfação dos médicos anteriormente dos testes e após. Serão também comparados o número médio de análises por dia antes e depois.

Se o tempo de processamento do sistema for demasiado elevado, a satisfação do utilizador final irá ser posta em causa.

Já no caso das configurações para os administradores de sistemas, serão medidos o número médio de *clicks* por configuração, juntamente com um índice de satisfação pela parte dos configuradores.

Em relação a segurança irão ser testados os ataques aos sistemas mais comuns (MITM no transporte de dados, vulnerabilidades comuns em websites como SQL Injections), verificando se o sistema se mantém operacional.

Segurança neste sistema é estritamente necessária, pois a solução lida com dados médicos sensíveis. O possível acesso a estes de forma ilícita tem de ser minimizado ao máximo. Também, a propriedade intelectual da FUJIFILM deve ser protegida (protocolos de comunicação, algoritmos, …).

7.2 Testes

Para testar o software criado para este projeto, foram elaborados os seguintes testes:

- Utilização de memória com 1 pipeline configurado
- Utilização de memória com 3 pipelines configurado
- Utilização de CPU com 1 pipeline configurado
- Utilização de CPU com 3 pipelines configurado

7.2.1 Linha de base

Antes da execução dos testes, foram feitas leituras da utilização de CPU e memória do sistema a correr durante 5 horas sem a execução do serviço principal.

Os resultados encontram-se na Tabela 7-1 – Resultados linha de base.

Tabela 7-1 – Resultados linha de base

Hora	1	2	3	4	5
CPU (Percentagem Média)	1.1	1.1	1.2	1.1	1.1
RAM (Utilização Média (MB))	35	35	35	35	35

Pode-se determinar que o sistema ARTIK 530 é estável perante um tempo de 5 horas, tempo que irá ser usado nos seguintes testes.

7.2.2 Testes de 1 pipeline

Para o teste com apenas 1 pipeline, a configuração mais comum foi usada, com a exceção do plugin Injector utilizado apenas para simular a entrada de dados:

- Plugins de entrada:
    - Injector – configurado para enviar dados de 500 em 500 milisegundos.
- Plugins de processamento:
    - AG1
    - HL7
- Plugin de output:
    - WebAPI

Resultando no seguinte pipeline: Injector -> AG1 -> HL7 -> WebAPI

Os resultados encontram-se na Tabela 7-2 – Resultados do teste com 1 pipeline.

Tabela 7-2 – Resultados do teste com 1 pipeline

Hora	1	2	3	4	5
CPU (Percentagem Média)	1.4	1.4	1.4	1.4	1.4
RAM (Utilização Média (MB))	43	43	43	42	43

7.2.3 Testes com 3 pipelines

Para este teste foi utilizada a configuração do teste com 1 pipeline, no entanto triplicada.

Os resultados encontram-se na Tabela 7-3 – Resultados do teste com 3 pipelines.

Tabela 7-3 – Resultados do teste com 3 pipelines

Hora	1	2	3	4	5
CPU (Percentagem Média)	2.0	1.9	2.0	2.0	1.9
RAM (Utilização Média (MB))	49	49	49	49	49

7.2.4 Conclusão

Para a chegada a uma conclusão com um valor significativo seria necessário a repetição destes testes várias vezes em várias circunstâncias. No entanto, a atividade de testes teve de ser interrompida até o aval do projeto pela parte da empresa principal.

Apesar destes testes rudimentares, utilizando o conhecimento intrínseco do código a ser executado, pode-se afirmar que a estabilidade nestes resultados é devido à alocação de memória ser feita apenas uma vez em casos de receção de dados num intervalo em que permita o processamento destes sem existir backlog. Isto permite resultados estáveis e sem muita flutuação de números.

Todos os resultados enviados para o WebAPI foram passados por uma ferramenta de validação de HL7, avaliando assim os resultados em 100% de confiança de dados corretos.

8 Trabalho Futuro

Este projeto foi desenhado como um sistema em constante evolução – a capacidade de aumentar a qualidade do produto e as funcionalidades deste é grande, como por exemplo:

- Plugins
    - Plugin de receção/envio de dados *raw* diretamente por TCP
    - *Update* de plugins para suportar sistemas de tentativa, erro e *backlog*.
    - Plugin de conversão para DICOM
    - Plugin de monitorização de ambiente utilizando *probes* (entrada de dados por I2C).

Todas estas modificações podem ser feitas num sistema já em produção pois são compatíveis com os sistemas anteriores e não necessitam de modificação do sistema principal que chama os próprios plugins.

No entanto, antes do desenrolamento do desenvolvimento é necessário testar mais afincadamente o sistema – estes testes estão programados para serem desenvolvidos mal exista a aceitação do sistema por parte da empresa principal.

9 Conclusão

Este projeto está a ser um projeto revolucionário no mundo da conectividade entre equipamentos médicos.

O sistema tem uma potencialidade enorme para a facilitação do acesso a dados pelos médicos e a facilitação da gestão de todo o tipo de análises e exames num hospital.

Infelizmente, sem o aval da empresa principal, todo o progresso no projeto (design, implementação e testes) teve de ser parado.

Como tema de dissertação, muitos dos pontos que seriam de interesse de discussão tiveram de ser retirados por questões de propriedade intelectual da empresa. Isto leva a uma experiência de leitura inferior.

Apesar destes fatores, penso que esta dissertação está ainda assim interessante e com dados que podem ser potencialmente aproveitados para a arquitetura de outras aplicações futuras.

10 Referências

Codd, E. F. (Junho de 1970). A Relational Model of Data for Large Shared Data Banks. San Jose, California, United States of America. Obtido em 13 de Fevereiro de 2017, de https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

Corepoint Health. (2009). *Corepoint Health.* Obtido em 10 de 10 de 2017, de Corepoint Health: https://corepointhealth.com/wp-content/uploads/hl7-history-v2-v3.pdf

FUJIFILM. (25 de Junho de 2014). *Immunodiagnostics Testing now available from Fujifilm.* Obtido em 20 de Fevereiro de 1027, de FUJIFILM Europe: https://www.fujifilm.eu/uk/news/article/immunodiagnostics-testing-now-available-from-fujifilm

FUJIFILM. (s.d.). *FUJIFILM NX500 Brochure.* Obtido em 20 de Fevereiro de 2017, de FUJIFILM Europe: https://www.fujifilm.eu/fileadmin/countries/europe/products/Medical/Brochures/NX500E_.pdf

FUJIFILM Holdings Corporation. (2016). *Annual Report 2016.* FUJIFILM Holdings Corporation. Obtido em 10 de Fevereiro de 2017, de http://www.fujifilmholdings.com/en/investors/annual_reports/2016/pack/pdf_TOP/Annual-Report-2016.pdf

Global Harmonization Task Force. (9 de Fevereiro de 2007). *Documents*. Obtido em 10 de Fevereiro de 2017, de International Medical Device Regulators Forum: http://www.imdrf.org/docs/ghtf/archived/sg1/technical-docs/ghtf-sg1-n045r12-in-vitro-diagnostic-classification-070209.pdf

Health Level Seven. (2003). *Transport Specification: MLLP, Release 1.* (R. Spronk, Ed.) Obtido em 10 de 10 de 2017, de Health Level Seven: http://www.hl7.org/documentcenter/public_temp_8402C3A8-1C23-BA17-0C4FEDACC1B55911/wg/inm/mllp_transport_specification.PDF

Health Level Seven. (10 de 10 de 2017). *HL7*. Obtido de HL7: http://www.hl7.org/about/FAQs/index.cfm?ref=nav

Intermedical. (s.d.). *Intermedical - FUJIFILM AG1.* Obtido em 20 de Fevereiro de 2017, de http://www.intermedical.jp/eng/images_up/goods/1782/pict2.jpg

ITU-T - Telecommunication standerization sector of ITU. (2012). *Overview of the Internet of things.* ITU. Obtido em 20 de Fevereiro de 2017, de http://www.itu.int/rec/T-REC-Y.2060-201206-I

Mikinaga Mori, J. K. (27 de Fevereiro de 2012). Development of Highly Sensitive Immunochromatographic Detection Kit for Seasonal Influenza Virus Using Silver Amplification. 5-10. Obtido em 10 de Fevereiro de 2017, de http://www.fujifilm.com/about/research/report/057/pdf/index/ff_rd057_all_en.pdf

Mimi Gentz, C. R. (22 de 11 de 2016). NoSQL vs SQL. Obtido de https://docs.microsoft.com/en-us/azure/documentdb/documentdb-nosql-vs-sql

Radiometer. (s.d.). *AQURE Enterprise – open, smart, integrated.* Obtido em 20 de Fevereiro de 2017, de Radiometer: http://www.radiometer.com/en/products/poc-management/aqure-enterprise-poc-management

Ringhold Whitepaper. (16 de 11 de 2007). *HL7 Message examples: version 2 and version 3*. (R. Spronk, Editor) Obtido em 10 de 10 de 2017, de Ringhold Whitepaper: http://www.ringholm.com/docs/04300_en.htm

Samsung. (2016). *ARTIK Modules 5.* Obtido em 13 de 10 de 2017, de ARTIK Modules 5: https://media.digikey.com/pdf/Data%20Sheets/Samsung%20PDFs/ARTIK-530_DS_V1.0_1-20-17.pdf

Samsung. (s.d.). *Samsung ARTIK™ 530/530s*. Obtido em 13 de 10 de 2017, de Samsumg ARTIK: https://www.artik.io/modules/artik-530/

Seven, H. L. (2010). *HL7: Version 2 Standard.* Obtido em 10 de 10 de 2017, de HL7: https://www.hl7.org/documentcenter/public_temp_AF0CBE3C-1C23-BA17-0C9CCAA1D5045DBB/calendarofevents/himss/2011/Version%202.pdf

Siemens. (2016). *Siemens Healthineers acquires Conworx Technology GmbH to deliver open connectivity for 100+ point-of-care instruments.* New York. Obtido em 20 de Fevereiro de 2017, de http://www.siemens.com/press/pool/de/pressemitteilungen/2016/Healthcare/PR2016110058HCEN.pdf