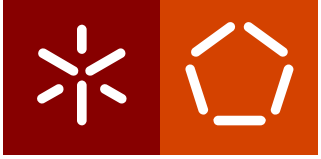
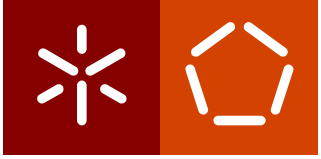


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Bruno Manuel Borlido Arieira

Indicadores de Desempenho em Plataformas HL7

Março 2021



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Bruno Manuel Borlido Arieira

Indicadores de Desempenho em Plataformas HL7

Master dissertation
Integrated Master's in Informatics Engineering

Dissertation supervised by
Professor Doutor José Manuel Ferreira Machado
Professor Doutor Hugo Daniel Abreu Peixoto

Março 2021

Despacho RT - 31 /2019 - Anexo 3

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Despacho RT - 31 /2019 - Anexo 4

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

AGRADECIMENTOS

Mais uma etapa se conclui ao longo de todo o meu percurso académico, e com a mesma surge uma necessidade imensa de expressar vários agradecimentos a todos aqueles que de uma forma ou de outra contribuíram para a realização da presente dissertação.

Em primeiro lugar, agradeço ao meu orientador, Professor Doutor José Machado, pela sua atenção e disponibilidade concedidas. Ao meu coorientador, Professor Doutor Hugo Peixoto, por todo o conhecimento transmitido, acompanhamento e dedicação, que mesmo com o tempo limitado se prontificou a impulsionar a presente dissertação, estando sempre recetivo a responder às minhas dúvidas. O meu sincero agradecimento por tudo.

Um eterno agradecimento aos meus avós, principalmente ao meu avô Luís, por ser um exemplo a seguir e por me ensinar que nunca se deve desistir dos objetivos traçados. Sem ele nenhuma das minhas concretizações a nível profissional teria sido possível.

Aos meus pais, pela oportunidade, por todo carinho e principalmente por estarem sempre presentes em todos os momentos, por nunca me deixarem desanimar e principalmente por acreditarem em mim. À minha irmã, pelo incentivo, apoio e colaboração nos momentos de incerteza.

Gostaria também de agradecer a todos os meus amigos que estiveram presentes no meu percurso académico, com quem fiz verdadeiros laços de amizade que levo comigo para a vida e que me acompanharam nos grandiosos, nos espetaculares, e como em tudo, nos menos bons momentos. Agradeço todas essas brutais aventuras pioneiras e às horas felizes que aqui passamos juntos.

Um grande obrigado aos meus amigos da velha guarda do grupo "Perre, Paris e Londres"e "A Culpa é do Mauro", pelos fantásticos momentos passados que serviram para descontrair durante esta longa caminhada académica.

À Cristiana, por ter tornado os últimos três anos inesquecíveis. Por ter sido fulcral ao longo do meu percurso académico e por nunca me ter deixado desistir. Pelas palavras e apoio nos momentos de maior dificuldade. Por toda a compreensão demonstrada. Pela amizade, amor e carinho.

Por último, agradecer a todos os docentes que contribuíram para a minha formação e aprendizagem ao longo de todos os anos de universidade.

A todos, um muito obrigado.

RESUMO

Nos dias de hoje torna-se essencial garantir a disponibilidade de toda a informação que circula em unidades na área da saúde, com a finalidade de auxiliar os profissionais de assistência médica. Por conseguinte, não deve existir qualquer tipo de barreira inerente à comunicação entre os sistemas de informação, mesmo que estes sejam distintos quanto à sua implementação.

É neste contexto que surge a interoperabilidade, isto é, a capacidade de diferentes sistemas estabelecerem a comunicação e troca de dados entre si, de forma eficaz, sem esforço adicional por parte do utilizador. Este conceito é promovido pela utilização de padrões específicos para troca de informação no domínio da saúde, como por exemplo o *Health Level 7* (*HL7*), e de sistemas de *software*, que proporcionam a normalização e partilha dos dados entre todos os sistemas, designados plataformas de integração.

Tendo em conta o elevado volume de mensagens a que este tipo de ferramentas poderá estar exposto, é indispensável que estejam operacionais 24 horas por dia, sem que exista qualquer tipo de sobrecarga de mensagens que comprometam o seu desempenho. Surge assim o objetivo principal da presente dissertação, a aquisição de métricas relevantes que proporcionam a performance ideal dos motores de integração, promovendo um fluxo de mensagens contínuo sem qualquer tipo de quebra. O *software* utilizado como objeto de estudo designa-se *Mirth Connect* (também denominado *NextGen Connect*).

Embora este tipo de sistemas de integração seja amplamente utilizado nas unidades de saúde, existe pouco trabalho de investigação inerente à avaliação do desempenho de ferramentas de interoperabilidade neste domínio específico. Tendo em consideração que a comunicação entre sistemas utilizada pela maioria deste tipo de plataformas é suportada por soluções *Message Oriented Middleware* (*MOM*), foram utilizados indicadores relacionados com abordagens de filas de mensagens e com boas práticas de processamento de fluxo de dados. Deste modo, para além de se promover os casos de análise e avaliação do comportamento da plataforma conforme métricas consideradas, também permite aos utilizadores deste tipo de ferramentas de *software* deterem uma noção das circunstâncias onde retiram o melhor aproveitamento do seu funcionamento.

Palavras-chave: Interoperabilidade, *Software* de Integração, Fluxo de Mensagens, Indicadores de Desempenho

ABSTRACT

Nowadays it is essential to ensure the availability of all the information that circulates in health units, in order to help health professionals. Therefore, there should be no inherent barrier to communication between information systems, even if they are different in terms of their implementation.

It's in this context that interoperability arises, the ability of different systems to establish communication and exchange of data with each other effectively, without additional effort on the part of the user. This concept is promoted by the use of specific standards for the exchange of information in the field of health, such as *Health Level 7* (*HL7*), and software systems, which provide the standardization and sharing of data between all systems, called integration platforms.

Given the high volume of messages to which this type of tool may be exposed, it is essential that they are operational 24 hours a day, without any type of message overload that compromises their performance. Thus emerges the main objective of this dissertation, the acquisition of relevant metrics that provide the optimal performance of the integration engines, promoting a continuous message flow without any kind of breakage. The software used as the object of study is called *Mirth Connect* (also called *NextGen Connect*).

Although this type of integration systems is widely used in health units, there is low research work inherent in evaluating the performance of interoperability tools in this specific area. Taking into account that the communication between systems used by most of this type of platforms is supported by *Message Oriented Middleware* (*MOM*) solutions, indicators related to message queue approaches and good data flow processing practices were used. Thus, in addition to promoting the cases of analysis and evaluation of the behavior of the platform according to the metrics considered, it also allows users of this type of software tools to have a notion of the circumstances where they take the best advantage of its operation.

Keywords: Interoperability, Integration Software, Message Flow, Performance Indicators

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

SIGLAS E ACRÓNIMOS

A

ACK Acknowledgement.

ADT Admission, Discharge and Transfers.

ANSI American National Standards Institute.

API Application Programming Interface.

ASCII American Standard Code for Information Interchange.

ATSM American Society for Testing and Materials.

C

CCOW Clinical Context Object Workgroup.

CDA Clinical Document Architecture.

CPU Central Process Unit.

CSS Cascading Style Sheets.

CSV Comma-separated values.

D

DFT Detail Financial Transactions.

DICOM Digital Imaging and Communications in Medicine.

DSR Design Science Research.

E

EDI Electronic Data Interchange.

ESB Enterprise Service Bus.

EVN Event Type.

F

FHIR Fast Healthcare Interoperability Resources.

FIFO First In First Out.

FTP File Transfer Protocol.

H

HDD Hard Disk Drive.

HIE Health Information Exchange.

HIMSS Healthcare Information and Management Systems Society.

HIT Health Information Technology.

HL7 Health Level 7.

HTML HyperText Markup Language.

HTTP HyperText Transfer Protocol.

HTTPS HyperText Transfer Protocol Secure.

I

IDE Integrated Development Environment.

IEEE Institute of Electrical and Electronics Engineers.

IMAP Internet Message Access Protocol.

IP Internet Protocol.

ISO Internacional Organization for Standardization.

J

JDBC Java Database Connectivity.

JMS Java Message Service.

JSON JavaScript Object Notation.

K

KEG Knowledge Engineering Group.

L

LCIM Levels of Conceptual Interoperability Model.

LIS Laboratory Information System.

LISI Levels of Information Systems Interoperability.

LLP Lower Layer Protocol.

M

MLLP Minimum Lower Layer Protocol.

MOM Message Oriented Middleware.

MSH Message Header.

MVC Model-View-Controller.

N

NACK Negative Acknowledgement.

NCPDP National Council for Prescription Drug Programs.

NOSQL Not Only SQL.

O

ODBC Open Database Connectivity.

ORM Order Message.

ORU Observation Result.

OSI Open System Interconnection.

P

PAS Patient Administration Systems.

PCE Processo Clínico Eletrónico.

PDF Portable Document Format.

PID Patient Identification.

PL Procedural Language.

POP3 Post Office Protocol.

PV1 Patient Visit.

R

RAM Random Access Memory.

RDF Resource Description Framework.

REST Representational State Transfer.

RIM Reference Information Model.

RTF Rich Text Format.

S

SFTP SSH File Transfer Protocol.

SGBDR Sistema de Gestão de Bases de Dados Relacional.

SI Sistemas de Informação.

SIH Sistemas de Informação Hospitalar.

SMB Server Message Block.

SMTP Simple Mail Transfer Protocol.

SOA Arquitetura Orientada a Serviços.

SOAP Simple Object Access Protocol.

SQL Structured Query Language.

SSD Solid-State Drive.

T

TCP Transmission Control Protocol.

TI Tecnologias de Informação.

TIC Tecnologias de Informação e Comunicação.

U

UML Unified Modeling Language.

URL Uniform Resource Locator.

W

WEBDAV Web-based Distributed Authoring and Versioning.

X

XML Extensible Markup Language.

Y

YAML YAML Ain't Markup Language.

1

INTRODUÇÃO

A corrente dissertação representa todo o trabalho desenvolvido no estudo e análise da plataforma *Mirth Connect* com o objetivo de obter o melhor desempenho possível para a mesma. Este estudo decorre no âmbito do último ano do Mestrado Integrado em Engenharia Informática, na Universidade do Minho.

Neste primeiro capítulo é realizada uma contextualização, dando principal destaque à importância da presença de plataformas de interoperabilidade na área da saúde e uma breve exposição da plataforma que irá ser objeto de trabalho. De seguida, são apresentadas as motivações que deram origem a esta dissertação, os objetivos a atingir para a concretização deste trabalho e, por fim, a forma pela qual este documento está organizado.

1.1 CONTEXTUALIZAÇÃO

Com o passar dos anos, na área da saúde acabou por se tornar obsoleto o método tradicional de se registar toda a informação em papel, sobretudo devido à enorme quantidade, complexidade e heterogeneidade de dados gerados. O fluxo eletrónico de informação digitalizada neste domínio, tem bastantes implicações para os processos clínicos e administrativos, além da privacidade e confidencialidade [10]. Para tal, por forma a combater estes aspetos, surgiu a necessidade de recorrer a *Sistemas de Informação (SI)*, ou seja, ao envolvimento de profissionais de *Tecnologias de Informação e Comunicação (TIC)* no domínio da saúde.

Os *Sistemas de Informação Hospitalar (SIH)* são sistemas responsáveis pela aquisição, processamento e apresentação de toda a informação acerca de todos os intervenientes (pacientes, médicos, enfermeiros, entre outros) e todos os serviços (clínicos, administrativos, entre outros) [11]. Este tipo de sistemas têm sido progressivamente um grande apoio para os profissionais de saúde, uma vez que promovem a acessibilidade e disponibilidade, assim como partilha das informações dos pacientes de forma simples e num curto espaço de tempo, promovendo uma redução no número e incidência de erros médicos, diminuição dos custos e espaço físico [12].

No entanto, tal como já foi referido, os SIHs encontram-se divididos por vários departamentos de maneira a auxiliar todas as infraestruturas dos mais diversos setores de uma

determinada instituição hospitalar [13]. Toda esta distribuição e diversidade relativamente a estes sistemas, obriga a que estes sejam geridos e administrados por forma a não perderem a sua integridade. Por exemplo, um paciente que esteja internado numa determinada instituição hospitalar, os profissionais de saúde da instituição em questão devem ter acesso à informação necessária do paciente (resultados de exames, medicação tomada, dados pessoais, etc) para que possam proceder com o tratamento de forma eficaz.

Ao longo dos anos, têm sido desenvolvidas imensas aplicações com o intuito de auxiliar os profissionais no domínio da saúde, no entanto, a maior parte apresenta um baixo nível na qualidade da troca de informação devido à forma individualizada de como esta é tratada, impossibilitando a comunicação entre outros sistemas. Assim sendo, é necessário incluir um elo intermediário que tenha a capacidade de estabelecer a interação entre diversos sistemas, mantendo a integridade e disponibilidade da informação, de modo a que haja cooperação entre eles [14]. Este requisito é o objetivo principal de infraestruturas *Health Information Exchange (HIE)*, que procuram facilitar o acesso e a integração do *Processo Clínico Eletrónico (PCE)*, de maneira a proporcionar uma assistência segura e eficaz do paciente, auxiliando os prestadores de serviços saúde [15].

Nestes termos, torna-se necessária a aplicação do conceito de interoperabilidade com o objetivo de promover a partilha das informações entre os serviços e evitar qualquer tipo de informação duplicada, omnipresente, heterogénea e distribuída. Deste modo, o principal objetivo é providenciar um sistema de informação homogéneo nas diversas unidades de saúde, com a capacidade de acolher qualquer tipo de informação viabilizando a sua integridade, confidencialidade, gestão, segurança e escalabilidade, procurando impossibilitar qualquer tipo de vulnerabilidade [16, 17].

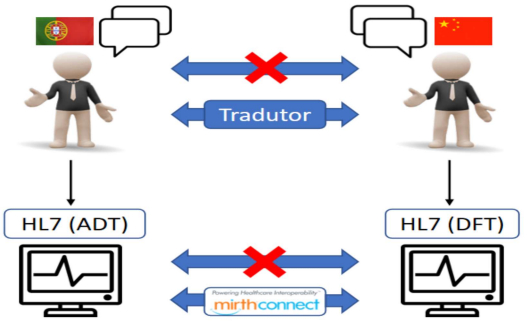


Figura 1.1: Exemplo de comparação.

Existem vários sistemas de *software* que são projetados e implementados sem ter em conta qualquer tipo de metodologia para a partilha integrada e interoperável dos dados. De modo

a entender o conceito desta abordagem, dá-se o exemplo de duas pessoas de diferentes nacionalidades que falam línguas diferentes e pretendem comunicar, recorrendo para tal a um tradutor que tenha a capacidade de interpretar os dois idiomas. Neste contexto, as ferramentas de integração e interoperabilidade agem com o mesmo sentido do tradutor e as pessoas como dois sistemas com diferentes semânticas e/ou protocolos (figura 1.1). Como já foi mencionado, aplicações na área da saúde são bastante úteis quando a sua utilização é realizada isoladamente, no entanto, também é necessário que a sua utilização seja efetuada com a finalidade de cooperarem e comunicarem com um vasto conjunto de sistemas com diferentes propósitos [18]. Neste contexto, a falta de comunicação é solucionada pela adoção de normas e metodologias, que seguem um conjunto de regras que permitem que as informações sejam compartilhadas e processadas de maneira uniforme e consistente. Entre os padrões existentes existem as normas *Health Level 7 (HL7)*, que garantem uma estrutura adequada e consistente para a partilha e integridade de informações de PCEs, proporcionando a interoperabilidade entre os SIHs [19].

Todavia, este tipo de padronização dos dados necessita de uma ferramenta que tenha capacidade de receber, enviar e tratar as mensagens que transportam a informação, de modo a estabelecer um fluxo de trabalho entre os inúmeros sistemas de assistência médica com diferentes formatações da informação. No âmbito deste trabalho, foi estudada em maior profundidade uma plataforma de integração capaz de estabelecer o fluxo de mensagens entre diversos sistemas, designada *Mirth Connect*. É uma plataforma *open-source*, que permite o envio bidirecional de mensagens HL7 entre sistemas, através da conexão estruturada por uma *Arquitetura Orientada a Serviços (SOA)*, possibilitando que as mensagens sejam filtradas, endereçadas ou transformadas com base em regras definidas pelo utilizador [20]. Este *software* atualmente designado *NextGen Connect*, é um dos líderes da indústria relacionada com soluções de interoperabilidade que segundo analistas de *Tecnologias de Informação (TI)* da área da saúde, *"levou todos os seus fornecedores a melhorar o acesso a dados externos (com um aumento de 27%) e prestação de cuidados (aumento de 25%)"* [21].

1.2 MOTIVAÇÃO

A existente multiplicidade de aplicações no âmbito da assistência médica leva a alguns desafios, dos quais se destacam a integração e a troca dos dados entre diversos sistemas. Para tal, para que se torne possível que exista partilha de informação, independentemente da sua heterogeneidade, de modo a constituir a comunicação e cooperação entre os sistemas, é necessário empregar o conceito de mecanismos de integração, de forma a garantir a interoperabilidade nas instituições de saúde.

Segundo a empresa *NextGen HealthCare*, onde se integra o *Mirth Connect*, tem um vasto número de clientes distribuídos por todo o mundo, aos quais fornecem este mecanismo de

integração [22]. Neste contexto, devido à importância inerente à informação hospitalar e ao seu acesso em tempo útil, esta plataforma está diariamente sujeita à receção e envio de um número elevado de mensagens que podem afetar o seu desempenho a nível de sobrecarga de mensagens, processando as mesmas num tempo não satisfatório.

O trabalho inerente a esta dissertação, insere-se num grupo de investigação denominado *Knowledge Engineering Group (KEG)*, que lida com o desenvolvimento e implementação de sistemas de informação na área da saúde. Deste modo, foram realizados trabalhos nomeadamente nas áreas de interoperabilidade, *data mining*, *business intelligence*, entre outras, identificados em [18][23][24][25]. Neste contexto, a possibilidade de melhorar a performance dos seus sistemas foi um fator relevante.

Desta maneira, este projeto de investigação permite a apresentação de métricas que aumentam o desempenho deste tipo de sistemas de *software*, propiciando aos utilizadores o conhecimento das circunstâncias onde retiram o melhor aproveitamento do seu funcionamento. Posto isto, através do desenvolvimento deste trabalho são proporcionados fluxos contínuos de mensagens, em tempo útil, para plataformas de integração, auxiliando os profissionais de assistência médica em termos de eficiência para a gestão do seu trabalho.

1.3 OBJETIVOS

Para proceder ao desenvolvimento deste trabalho é necessário ter em vista algumas metas que incentivem à solução das características referidas anteriormente. Para tal, é necessário reter alguns conceitos base que servirão de apoio à investigação desta plataforma, análise esta que será realizada através de testes com o auxílio de ferramentas de virtualização.

De modo a orientar o desenvolvimento deste trabalho, são definidas algumas questões de investigação que sustentam o plano estrutural deste estudo:

1. Quais os indicadores que retrataram uma boa performance para as plataformas de integração?
2. Qual a arquitetura ideal para a implementação de um sistema de integração baseado num motor de mensagens HL7?
3. As ferramentas de avaliação usadas são escaláveis e aplicáveis a outros motores de integração?

Posto isto, com o propósito de auxiliar as questões de investigação colocadas acima, os objetivos fundamentais para a realização desta dissertação encontram-se organizados da seguinte forma:

- Explorar as caraterísticas da norma HL7 e ferramentas de geração automática de mensagens;

- Estudo de plataformas de interoperabilidade clínica e interpretação do seu funcionamento;
- Análise e aquisição de indicadores de desempenho;
- Avaliação e investigação da performance na plataforma de estudo;
- Atingir a melhor estrutura lógica para performance e averiguar a sua aplicabilidade em ambiente real;
- Reconhecer e constatar medidas que possam ser tomadas por forma a melhorar o desempenho do *software*;

1.4 ESTRUTURA DO DOCUMENTO

Este documento, encontra-se organizado por sete capítulos: Introdução, Estado da Arte, Metodologia de Investigação, Plataforma de Integração - Mirth Connect, Tecnologias de Desenvolvimento, Arquitetura e Implementação e Conclusão. Relativamente a este primeiro capítulo, apresenta uma contextualização com o objetivo de enquadrar o leitor acerca da temática que se baseia esta dissertação, de seguida a motivação que leva ao desenvolvimento deste trabalho e os objetivos. De seguida, segue-se uma breve descrição dos restantes capítulos:

- **Capítulo 2 (Estado da Arte):** são apresentados os conceitos essenciais que dão suporte à realização desta dissertação, ou seja, o Estado da Arte. Assim, é explorado o conceito de interoperabilidade, e em continuidade, é aprofundado o conhecimento sobre a norma HL7. De seguida, é abordado o conceito de plataformas de integração, onde são apresentadas 3 ferramentas de um modo geral, e posteriormente é feita uma análise acerca do motor de integração mais adequado para o objeto de estudo deste projeto. Neste capítulo, é averiguado o conceito de monitorização e a definição de algumas medidas de desempenho relevantes. Por fim, são apresentados trabalhos relacionados, onde são descritas duas abordagens que representam métodos para propiciar o fluxo de mensagens em *softwares* de integração, filas de mensagens e processamento de fluxo de dados. São também retratadas métricas de desempenho de trabalhos de investigação de avaliação de *softwares* similares.
- **Capítulo 3 (Metodologia de Investigação):** neste capítulo é exposta a abordagem metodológica que foi utilizada, com a finalidade de organizar o documento escrito e trabalho desenvolvido. É também apresentada uma combinação das etapas inerentes à metodologia empregue e de todos os capítulos desta dissertação.

- **Capítulo 4 (Plataforma de Integração - *Mirth Connect*):** Aqui são apresentadas todas as noções acerca do motor de integração que é objeto de estudo de toda esta investigação. Deste modo, este capítulo demonstra o resultado obtido do estudo e pesquisa deste sistema de *software*, desde a sua arquitetura até a maior parte de todas as suas funcionalidades, para posteriormente facilitar a sua utilização aquando a sua análise.

- **Capítulo 5 (Tecnologias de Desenvolvimento):** onde são expostas todas a tecnologias e ferramentas necessárias para o desenvolvimento deste projeto. Para cada ferramenta é feita uma breve apresentação, assim como uma justificação acerca do motivo que levou à necessidade da sua utilização.

- **Capítulo 6 (Arquitetura e Implementação):** neste capítulo são abordados os aspetos principais que complementam este trabalho. Inicialmente foi apresentada a arquitetura montada no *Docker* para criação de um ambiente de teste e, de seguida, foi explicada a arquitetura de canais utilizada no *Mirth Connect* que proporcionou a análise das mensagens. Seguidamente, são expostos os indicadores que geram alterações ao nível do fluxo de mensagens e por fim, é feita uma avaliação dos resultados obtidos.

- **Capítulo 7 (Conclusão e Trabalho Futuro):** retrata as principais conclusões inerentes a esta investigação e ainda é feito um resumo de todo o trabalho realizado. São também apresentadas respostas a questões de investigação e, por fim, é feita uma análise dos pontos que poderiam ser tratados como trabalho futuro, de forma a facilitar a monitorização de plataformas de integração.

2

ESTADO DA ARTE

Neste capítulo são apresentados de forma mais pormenorizada os conceitos teóricos fundamentais que servem de auxílio a este estudo. Sendo a plataforma *Mirth Connect* o objeto de estudo, torna-se essencial especificar os conceitos de interoperabilidade e os seus *standards*, como a norma HL7. São também apresentados e relacionados *softwares* de integração, por forma a averiguar os que constituem as caraterísticas ideais para proceder á sua análise durante este trabalho.

De seguida, é abordada a temática de monitorização e medidas de desempenho normalmente utilizadas, que será relevante no sentido da avaliação e acompanhamento do sistema. Por fim, são apresentados alguns trabalhos relacionados, sendo retratadas abordagens que constituem boas práticas para promover a performance de ferramentas de integração e métricas de desempenho a ter em conta durante a monitorização deste tipo de sistemas.

2.1 INTEROPERABILIDADE

Nos dias de hoje, fornecedores na área da saúde impuseram serviços de TI nos seus fluxos de trabalho diários, com determinado grau de independência [18]. Esta independência implícita, pode ser a causa de dificuldade na interoperabilidade entre SI pois a sobrecarga deste tipo de sistemas numa unidade de saúde pode levar a problemas no acesso ao total de informações necessárias, ou seja, é difícil para um médico aceder a todas as fontes de informação num período de tempo aceitável. Segundo uma publicação da *Healthcare Information and Management Systems Society (HIMSS)*, *"Atualmente a falta de interoperabilidade compromete a segurança do paciente; afeta a qualidade do atendimento e os resultados e, em última análise, desperdiça biliões de dólares todos os anos"*[26], o que apoia plenamente a importância deste conceito.

Em conformidade com a definição do *Institute of Electrical and Electronics Engineers (IEEE)*, a interoperabilidade é *"a capacidade de dois ou mais sistemas ou componentes, trocarem informações e usar informações que foram trocadas"*[27]. Uma outra definição, segundo a *Internacional Organization for Standardization (ISO)*, é *"a capacidade de comunicar, executar programas ou transferir dados entre várias unidades funcionais de uma maneira que exija que o utilizador tenha*

pouco ou nenhum conhecimento das características exclusivas dessas unidades"[28]. Assim sendo, pode-se inferir de forma introspetiva que interoperabilidade é a capacidade de diferentes sistemas conseguirem estabelecer a devida comunicação e partilha de dados entre si de forma eficaz, confiável e consistente, sem qualquer tipo de esforço adicional por parte do utilizador.

Cross-Domain Interoperability (interoperabilidade entre domínios) é fundamental pois possibilita aos sistemas e organizações a interação e troca de informações (interoperar) entre diferentes áreas, mercados, indústrias, etc [29]. Por exemplo, numa situação de um desastre natural é indispensável haver partilha de informações entre todas as organizações intervenientes (ou domínios), de forma a promover a comunicação e coordenação das suas ações, para atingir os seus objetivos. Neste âmbito, como já foi mencionado no capítulo anterior, este conceito é bastante importante na área da saúde, em que os dados são relativos a pacientes, sendo primordial manter o fluxo de dados entre qualquer tipo de sistema interoperável neste domínio, pois o bem-estar e saúde dos doentes é determinante.

Existem vários modelos que permitem a organização e classificação de sistemas de informação relativamente ao latente nível de interoperabilidade. Não obstante, alguns salientam-se devido a terem em consideração alguns atributos que classificam os sistemas a nível da troca de informações. Deste modo, destaca-se um modelo ultimamente atualizado por *Tolk* e *Wang's* [1], designado de *Levels of Conceptual Interoperability Model (LCIM)*, que tem principal finalidade de representação dos níveis de interoperabilidade de determinados sistemas.

A figura 2.1 retrata o modelo anteriormente referido e os sete níveis de interoperabilidade definidos. Procede-se as características inerentes a cada nível [30]:

- **Nível 0:** É caraterizado por sistemas independentes, com interoperabilidade inexistente.
- **Nível 1:** Na **interoperabilidade técnica** é utilizado um protocolo de comunicação entre sistemas, que permite a troca de informação por intermédio da rede e de protocolos bem definidos. Desta forma, neste nível existe a concordância entre os conceitos *plug and play* (conecte e use), sinal e protocolo.
- **Nível 2:** Relativamente à **interoperabilidade sintática**, está integralmente associada ao formato dos dados, isto é, é introduzida uma estrutura comum para o intercâmbio da informação entre sistemas. Assim, um protocolo comum é utilizado para estruturação dos dados, onde a padronização para a troca de informações é definida sem ambiguidade.
- **Nível 3:** A **interoperabilidade semântica** refere-se à capacidade de dois ou mais sistemas interpretarem automaticamente as informações trocadas de forma significativa

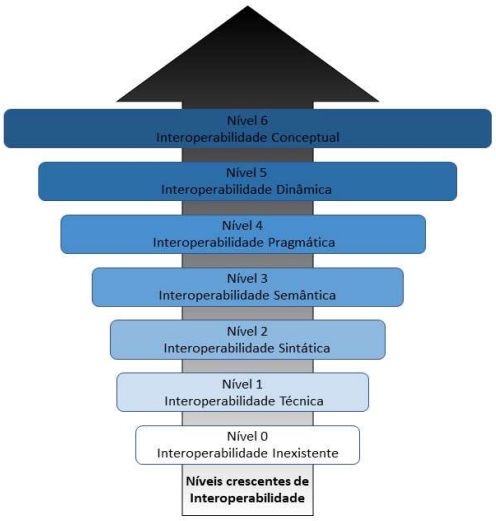


Figura 2.1: Modelo LCIM (adaptado de *Tolk* e *Wang's* [1]).

e precisa, a fim de produzirem resultados úteis, conforme o definido pelos utilizadores. Neste âmbito, fornece a garantia de que o sistema recetor compreenda o significado da informação trocada, mesmo quando os algoritmos usados pelo sistema recetor sejam distintos em relação aos usados pelo sistema emissor.

- **Nível 4:** De modo a obter a **interoperabilidade pragmática** é necessário que os sistemas intervenientes tenham consciência dos procedimentos ou métodos que cada sistema efetua. Isto implica que a troca dos dados ou o contexto onde esta partilha acontece, seja compreendido pelos sistemas participantes.

- **Nível 5:** À medida que os sistemas vão operando sob os dados ao longo do tempo, o estado desses sistemas vão sendo alterados. Diz-se que um sistema atingiu a **interoperabilidade dinâmica** quando os sistemas são capazes de perceber as mudanças de estado e tirar proveito das mesmas. O efeito causado das trocas de informação é definido de maneira inequívoca para todos os sistemas participantes.

- **Nível 6:** De maneira a atingir o maior nível de **interoperabilidade conceptual** é fundamental que os sistemas estejam de acordo conforme as premissas e restrições latentes em cada ambiente real. Para tal é necessário documentar os modelos conceptuais através de métodos utilizados na engenharia, para que qualquer profissional de TI

seja capaz de os compreender. Quando este tipo de interoperabilidade é alcançada, os sistemas intervenientes podem ser aplicados a ambientes distintos onde as premissas e restrições são discordantes.

Como se pode verificar pelo modelo LCIM acima apresentado na figura 2.1, conforme os níveis vão aumentando, a capacidade de interoperação entre os sistemas também expande. Um outro aspeto pertinente que se pode observar, pela interpretação dos diferentes níveis, é que o último nível conserva todas as caraterísticas dos anteriores, assim como o nível 5 engloba todos os outros níveis prévios. Desta perspetiva, caso o nível 4 seja atingido, significa que todas as caraterísticas dos níveis precedentes, tal como as do próprio nível, também foram atingidas [30]. Desta forma, o significado desta interpretação é que quando se alcança a interoperabilidade conceptual, considera-se que todas as particularidades dos níveis inferiores foram atingidas pelos sistemas, o que simboliza que se atingiu o nível ótimo de interoperabilidade. No entanto, não significa que o último nível seja o mais adequado pois esse fator é determinado pelos requisitos de cada sistema de informação.

O modelo anteriormente referido foi inicialmente construído por *Andreas Thor* com base numa abordagem dos *Levels of Information Systems Interoperability (LISI)* [30]. Esta noção teve como influência um dado modelo de maturidade, que define uma base comum do Departamento de Defesa dos Estados Unidos para a definição de requisitos e para melhorias incrementais do sistema. Neste contexto, os sistemas amadurecem a partir de estágios, a fim de melhorar suas chances de interoperar com outros sistemas. Para a modulação os diferentes estágios para obter a máxima maturidade ao longo do tempo, dividiram em três instâncias [31]:

1. **Integridade :** Processo de configurar e modificar um conjunto de componentes para torná-los interoperáveis. Pode começar por ser denotado pelos níveis 1 e 2 do modelo LCIM, onde os sistemas começam-se por começar a adaptar às especificidades do meio onde se inserem.

2. **Interoperabilidade :** Capacidade de determinado sistema estabelecer a comunicação, troca e compreensão de qualquer tipo de informação com outros sistemas, por forma a conseguirem cooperar conforme determinado objetivo. Começa-se por evidenciar no nível 3, onde já existe a interpretação dos dados trocados, independentemente dos seus formatos.

3. **Composabilidade :** Mais do que ter a capacidade de interoperar com outros sistemas é conseguir analisar e perceber a estrutura onde se insere determinado sistema e conseguir adaptar-se à arquitetura. Esta dimensão começa por se observar no nível 4 e 5, onde os sistemas se vão adaptando aos diferentes estados.

A existência de interoperabilidade nas mais diversas instituições de saúde proporcionam **benefícios** que são essenciais e preponderantes, dos quais destacam-se [32]:

- **A melhoría na experiência do paciente:** muitas vezes cabe ao paciente ter de levar os resultados de exames de uma instituição para outra ou até fazer o pedido para alterar alguns dados que se encontram desatualizados. Estas situações não acontecem quando existem sistemas interoperáveis que permitem o fluxo contínuo de informação.
- **A redução significativa dos custos:** sistemas de saúde financiam docentes com a tarefa de inserir, eliminar ou editar dados nos diferentes sistemas, o que torna um processo lento, dispendioso e com alta probabilidade de ocorrência de erros. Com a aplicação desta abordagem, melhora a qualidade do trabalho e promove uma contenção de custos.
- **O alcance de melhores resultados:** se todos os prestadores de cuidados tivessem a vantagem de obterem dados completos e íntegros nos diversos pontos de atendimento para uma melhor avaliação dos seus pacientes, possibilitava uma tomada de decisão mais concisa, o que melhorava o nível de atendimento e evitava a ocorrência de possíveis erros, que poderíam ser fatais.

Como em todas as abordagens, existem desafios e adversidades que necessitam de ser ultrapassados no âmbito da interoperabilidade. Segundo estudos realizados em 2017 [33], nos Estados Unidos apenas 30% dos hospitais conseguiram satisfazer as quatro métricas principais para se obter uma interoperabilidade de excelência (integração, localização, distribuição e receção de dados), sendo que a percentagem referida anteriormente foi a subida do valor 24,5% de passados três anos, significando que um em cada 20 hospitais alcançou interoperabilidade nesse período. Estes valores não são necessariamente por culpa dos fornecedores de TI na área da saúde, pois até há relativamente pouco tempo as instalações não eram realizadas de modo a tornar a interoperabilidade como uma prioridade. Apenas recentemente, numa era onde o atendimento é realizado com base em valores e em que todos os aspectos de uma organização estão sob avaliação, é que os administradores começam a repensar se os sistemas que eles implementam, atuam como um obstáculo ao atendimento geral [34].

Neste contexto, é importante salientar explicitamente algumas das **barreiras** da implantação desta noção. Deste modo, um dos principais desafios da implementação da interoperabilidade é a incapacidade de reconhecimento de pacientes e a falta de correspondência dos seus registos nas configurações dos prestadores de serviços de saúde, onde a incompatibilidade de registos de pacientes pode levar a erros no atendimento e aumentar a probabilidade de danos ao paciente. Isto deve-se ao facto da **inexistência de um identificador nacional de paciente** num sistema PCE, pois em muitos países é revogado devido a políticas de privacidade [35].

Uma outra barreira relaciona-se com os **custos elevados em termos da integração**, onde os provedores de assistência médica utilizam vários sistemas que permitem localizar e

analisar dados relevantes para integrar a informação de forma eficaz. Para tal, é preciso a existência de conetividade e integração entre um número elevado de diferentes sistemas de PCE e *Health Information Technology (HIT)*, que tem como consequência custos significativos onde apenas empresas de maior calibre tem possibilidade para suportar estes gastos.

Uma outra adversidade considerada, bastante importante nesta abordagem é a **padronização da informação**. Como já foi abordado anteriormente (no exemplo representado pela figura 1.1), nas mais diversas instituições hospitalares existem inúmeros sistemas de informação com diferentes protocolos de comunicação e que cuja representação da informação é variável, assim como o formato da mesma. Toda este heterogeneidade, leva a que estes sistemas com PCEs se tornem incapazes de comunicar com outros que possuam protocolos distintos. Por conseguinte, esta falta de padrões de interoperabilidade pode obstruir a troca contínua de dados de saúde, complicando as transações e colocando barreiras adicionais ao fluxo de informações [36]. No entanto, este desafio com o passar do tempo, cada vez mais tem sido ultrapassado devido à imposição de plataformas de interoperabilidade (ou ferramentas de integração), como já se conferiu no capítulo anterior.

2.2 HEALTH LEVEL 7

Com o aumento do número de soluções informatizadas na área da saúde, a necessidade de partilhar dados clínicos de maneira transparente entre sistemas tornou-se essencial. De forma a responder a esta primordialidade, especialistas na setor das Tecnologias de Informação reuniram-se, e em 1987 fundaram uma organização designada de HL7 *International* sem fins lucrativos e certificada pela *American National Standards Institute (ANSI)* em 1994, com membros em mais de 50 países [37]. Esta fundação tem como principal finalidade o desenvolvimento de padrões para auxiliar os prestadores de serviços de saúde a transferir dados clínicos e administrativos de maneira uniforme, entre sistemas de *software*.

A alusão ao nível 7 advém do facto dos vários padrões se concentrarem na sétima camada (última) do modelo de comunicações para interconexão entre sistemas (*Open System Interconnection (OSI)*) - a camada de aplicação. Esta camada corresponde a aplicações que são utilizadas de modo a promover uma interação homem-máquina.

Tendo em conta que a norma possui a mesma designação que a própria organização, a HL7 é também um protocolo internacional de mensagens que facilita o intercâmbio de grandes volumes dados eletrónicos em todos os ambientes da área da saúde, integrando informações de natureza clínica e administrativa [38].

Uma vez que este intercâmbio de informação é de natureza hospitalar, pode se relacionar o *standard* HL7 com o conceito de HIE já abordado. Isto torna-se possível porque ambas as abordagens tem como principal propósito de promover a troca de informação, assegurar a interoperabilidade. Por outras palavras, para alcançar a interoperabilidade entre SIHs são

desenvolvidas normas, como a HL7 que possibilita a interpretação por todos os sistemas envolvidos, providenciando desta maneira o intercâmbio de informação.

Segundo a organização HL7 *International* existem padrões que são correntemente utilizados e implementados. Desta forma, de muitos existentes destacam-se padrões como [39]:

- **Versão 2:** também designado de *pipehat* (devido à terminologia usada nas suas mensagens), é constituída por várias versões e tem como principal objetivo de oferecer suporte aos fluxos de trabalho inseridos em instituições hospitalares através de mensagens estruturadas conforme uma morfologia bem delineada, como se poderá observar mais à frente. As mensagens são conhecidas por serem um pouco complexas, devido à sua estruturação em *ASCII* e aos diversos componentes inerentes a cada mensagem. Esta versão HL7 é principalmente utilizada em sistemas de administração de pacientes (*PAS*), sistemas de informações de laboratório (*LIS*), sistemas de registo eletrónico de saúde (PCE), sistemas dietéticos, farmácia e cobrança.

- **Versão 3:** devido à limitação de tecnologias já ultrapassadas e à existência de um elevado número de versões da norma HL7 v2, surgiu a necessidade de criação de algo mais atualizado. Desta forma surgiu esta versão, com intuito de atualizar a versão anterior, principalmente na área das tecnologias usadas e resolver muitos dos problemas existentes nessa versão. A principal diferença desta versão do *standard*, em relação à v2.x, é o facto de estar assente num modelo de referência de informação, o *Reference Information Model (RIM)* [40]. Este modelo é baseado na linguagem *UML*, consiste num conjunto de classes (ou *tags*) das quais derivam outras classes mais específicas. Por exemplo, subclasses da classe "ato" incluem "observação" e "procedimento". A utilização desta abordagem orientada a objetos, que se baseia numa sintaxe de codificação *XML*, tem como vantagem clarificar as definições da norma, garantir a consistência e consequentemente uma interoperabilidade semântica.

- ***Clinical Document Architecture (CDA)*** é baseado na versão anteriormente descrita, no entanto, a versão 3 relaciona-se com um sistema de mensagens e esta versão assenta num paradigma de documentos. A diferença está que no paradigma de documentos o objetivo principal é a legibilidade humana enquanto que nas mensagens a principal finalidade está presente no processamento da máquina [41]. É também baseado em *XML*.

- ***Clinical Context Object Workgroup (CCOW)*:** é um padrão vocacionado para a gestão de contexto clínico, projetado para, essencialmente, reduzir a sobrecarga administrativa e o fluxo de trabalho. Através desta gestão de contexto, concede que um utilizador obtenha uma visão unificada de um foco de interesse, como um paciente, agendamento de uma consulta, item de pagamento, etc.

– ***Fast Healthcare Interoperability Resources (FHIR)*:** tal como nas versões anteriores, tem como propósito principal a promoção da interoperabilidade nos sistemas. Todavia, este padrão descreve os formatos e elementos de dados (também designados de recursos), sendo mais fácil de implementar pois usa fundamentalmente tecnologias mais recentes (*web-based*), incluindo *HTTP*, *HTML* e *CSS* para incorporação da interface do utilizador e *JSON*, *XML* ou *RDF* para representação de dados [42]. Este é o *standard* HL7 mais recente e proporciona a exposição dos elementos de dados como serviços. Por exemplo, elementos básicos de assistência médica (pacientes, medicamentos, etc) podem ser recuperados e manipulados pelos seus *Uniform Resource Locators (URLs)* de recursos.

Depois de especificadas as versões e um pouco acerca da história do surgimento desta padronização, torna-se necessário entender a forma de como os dados são modificados em todas as plataformas quando ocorre um evento. Por exemplo, quando ocorre a admissão de um paciente num hospital é necessário que seja atualizado nos restantes sistemas para, a título de exemplo, a atualização de um registo de camas ocupadas.

Quando ocorrem este género de ocorrências é despoletado um ***trigger event***, definido como um conjunto explícito de condições que iniciam a transferência de informações entre os diversos componentes do sistema [43]. Este evento pode ser gerado por três motivos: por solicitação do utilizador, por mudança de estado ou com base na interação.

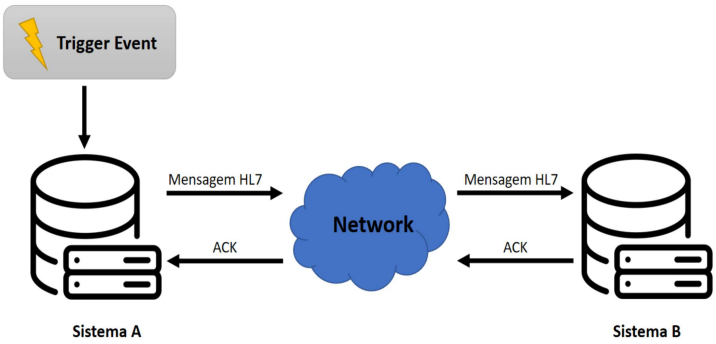


Figura 2.2: Exemplo da troca de uma mensagem HL7, despoletada por um *trigger event* (adaptado de [2]).

Na figura 2.2 está representado um exemplo de troca de mensagens entre dois sistemas, onde uma mensagem HL7 é enviada por uma entidade emissora (”Sistema A”), por reação ao *trigger event* , para o ”Sistema B” através da rede e a entidade recetora como confirmação da receção da mensagem HL7 envia um *ACK*.

2.2.1 *Caraterização de uma mensagem HL7*

Para a execução deste trabalho é necessário um estudo prévio da sintaxe das mensagens HL7, por exemplo, para o mapeamento de elementos de dados em variáveis para um possível armazenamento ou envio para uma determinada bases de dados ou para uma fonte recetora distinta. Como a norma **HL7 v2** será a utilizada neste trabalho de investigação e é a que apresenta a sintaxe mais complexa, será a que se vai analisar, por forma a clarificar as mensagens desta versão.

Cada uma das mensagens consiste em campos de dados com comprimento variável, que são separados com o uso de um carater de separação de campos e podem repetir-se. Os campos de dados são combinados em grupos lógicos chamados segmentos. Estes começam com o literal de três caracteres e podem ser definidos como opcionais ou necessários. Uma mensagem HL7 é constituída pelos seguintes componentes [44]:

- **Mensagem:** composta por um conjunto de segmentos que apresenta uma sequência definida. Dependendo do tipo da mensagem, é delineado um código de três carateres. Cada mensagem tem um tipo que define o seu objetivo. É esse tipo de mensagem que se traduz num código com três caracteres.

- **Segmentos:** grupo lógico de campos que define a única estrutura onde é possível criar diretamente uma mensagem. Podem ser definidos como opcionais ou obrigatórios, e podem ser repetidos numa mensagem. Cada um contém uma designação (código de três caracteres), que identifica o segmento e é chamado de *segment ID*. Os mais usados são: *MSH* - cabeçalho da mensagem; *PID* - identificação do paciente; *PV1* - visita do paciente; *EVN* - Tipo de evento, etc.

- **Campos:** são *strings* de carateres onde cada um pertence a um segmento específico. Tal com o anterior, pode ser obrigatório ou opcional e pode haver repetição. Cada campo possui um número de sequência que pode ser nulo. Estes tem a possibilidade de se dividir em subcampos e/ou subsubcampos e apresentam algumas propriedades:

  - Posição (POS)
  - Comprimento Máximo (LEN)
  - Tipo de dados (DT)
  - Opcionalidade (OPT)
  - Repetição (RP)
  - Tabela (TBL)
  - Número ID (ITEM)
  - Nome (ELEMENT NAME)

- **Delimitadores de Mensagem:** conjunto de carateres especiais cuja principal finalidade é a separação e reconhecimento de campos e segmentos já que as mensagens são um conjunto de carateres. Deste modo, existem 6 carateres especiais:

Designação	Função	Carater Especial
Fim de segmento	Termina um segmento.	<**cr**>
Separador de campos	Separa dois campos adjacentes dentro do segmento.	|
Separados de subcampos	Separa dois subcampos adjacentes, caso existam.	^
Separador de subsubcampos	Separa dois subsubcampos adjacentes, caso existam.	&
Separador de repetições	Separa múltiplas ocorrências do campo, caso existam.	~
Carater de escape	Utiliza-se quando é necessário representar um dos 6 carateres	\

Tabela 2.1: Especificação dos delimitadores de mensagens HL7 v2.

O *standard* HL7 possuí vários tipos de mensagem, dependendo do tipo de pedido efetuado. Desta maneira, distinguem-se quatro tipos de mensagens usualmente utilizados:

- ***Admission, Discharge and Transfers (ADT)*:** estas mensagens carregam informações importantes sobre *trigger events*, como a admissão de pacientes, transferência, registo, alta médica, entre outros.
- ***Order Message (ORM)*:** tipo de mensagem que contém informações acerca de um pedido, ou seja, é acionada quando os pedidos são criados, modificados, eliminados, colocados em espera, etc.
- ***Observation Result (ORU)*:** é usado como resposta a pedidos, enviando resultados de laboratório clínico, relatórios de estudos de imagem, resultados de exames imagiológicos, condição do paciente (como sinais vitais, alergias, entre outros), etc.
- ***Detail Financial Transactions (DFT)*:** Tal como o próprio nome sugere, este tipo de mensagens são usadas para descrever transações financeiras que são enviadas para sistemas de faturamento e para fins de contabilidade do paciente.

```
MSH|^~\&|GHH LAB|ELAB-3|GHH OE|BLDG4|200202150930||ORU^R01|CNTRL-3456|P|2.4<cr>
PID|||555-44-4444||EVERYWOMAN^EVE^E^^^^L|JONES|19620320|F|||153 FERNWOOD DR.^
^STATESVILLE^OH^35292||(206)3345232|(206)752-121||||AC555444444||67-A4335^OH^20030520<cr>
OBR|1|845439^GHH OE|1045813^GHH LAB|1554-5^GLUCOSE|||200202150730|||||||||
555-55-5555^PRIMARY^PATRICIA P^^^^MD^^|||||||||F||||||444-44-4444^HIPPOCRATES^HOWARD H^^^^MD<cr>
OBX|1|SN|1554-5^GLUCOSE^POST 12H CFST:MCNC:PT:SER/PLAS:QN||^182|mg/dl|70_105|H|||F<cr>
```

Figura 2.3: Exemplo de uma mensagem HL7 do tipo ORU-R01 (obtido de [3]).

A figura 2.3 reflete um exemplo de uma mensagem HL7 do tipo ORU. Tal como se pode observar, a mensagem tem implícita todos os constituíntes que foram mencionados anteriormente de forma detalhada.

2.3 PLATAFORMAS DE INTEGRAÇÃO

Qualquer ambiente médico no âmbito das Tecnologias de Informação, é composto por uma mistura de softwares, hardwares, protocolos, etc. Na maioria das infraestruturas hospitalares, as aplicações foram desenvolvidas com um propósito específico, sendo necessária a utilização da informação de pacientes.

Muitos destes *softwares* são desenvolvidos sem terem em vista a comunicação com outros já existentes, isto é, são implementados de forma independente. Quando existe um elevado número de sistemas e protocolos díspares a serem utilizados, é praticamente impossível de estabelecer qualquer tipo de interação e comunicação entre eles, sendo necessário a existência de uma ferramenta intermediária que torne possível estes aspetos - **motor de integração**.

As **plataformas (ou motores) de integração** são responsáveis pela normalização dos dados e pela introdução de informações entre todas as aplicações. Tem a capacidade de conetar sistemas internos e externos e geralmente é responsável por permitir melhorias no fluxo de trabalho, de que dependem as equipas de assistência médica. Estes mecanismos de integração são considerados como um ***Hub*** num ambiente de TI.

Estas ferramentas tem a aptidão de receber mensagens de sistemas distintos, de transformá-las em formatos que possam ser entendidos por sistemas recetores e, por fim, de distribuí-las. Este processo pode ser realizado em diferentes formatos, tais como HL7, FHIR e pedidos baseados em *HTTP*. Devido à padronização HL7 ser uma norma existente na globalidade destas plataformas e um *standard* fulcral no que toca à interoperabilidade, estas ferramentas também são designadas de **interfaces HL7**.

Cada vez mais aparecem novas ferramentas de integração na área da saúde e as existentes são melhoradas e atualizadas periodicamente, conforme novas demandas e necessidades das instituições hospitalares, sempre com principal propósito de garantir a interoperabilidade nas diversas infraestruturas. De seguida são apresentadas três plataformas de integração, classificadas por uma organização designada **KLAS *Research***, que se dedica a investigar e estudar as melhores soluções em termos de *softwares* na área da saúde no domínio da integração dos dados [45].

2.3.1 *Mirth Connect*

Tal como já foi falado anteriormente, esta dissertação baseia-se numa plataforma específica designada *Mirth Connect* que será o objeto de estudo durante todo o trabalho desenvolvido.

Mirth foi um produto criado por a empresa ***WebReach, Inc***, na Califórnia, e o lançamento da primeira versão surgiu em 2006 no *SourceForge*. Em 2009, a popularidade deste *software* era de tal forma que a própria empresa mudou de nome para ***Mirth Corporation*** e o seu

nome mudou para ***Mirth Connect***. Em Setembro de 2013, a empresa foi adquirida pelos sistemas de informação ***NextGen Healthcare*** [46].

O sistema de *software* ***NextGen Connect***, originalmente designado de ***Mirth Connect***, é um *middleware open-source* que permite a conexão e interação entre sistemas de informação no domínio da saúde. Desta forma, esta ferramenta proporciona a partilha e processamento de dados clínicos e administrativos em instalações médicas, estabelecendo assim o fluxo contínuo de informação numa dada infraestrutura [47].

Esta ferramenta também possibilita a transformação dos dados de um formato para outro pretendido ou até proceder à extração de determinados fragmentos dos dados, nos quais se pode optar por operar sobre eles ou enviar para um outro sistema. As interfaces que são configuradas para executar estes tipos de trabalhos são designadas de **canais**.

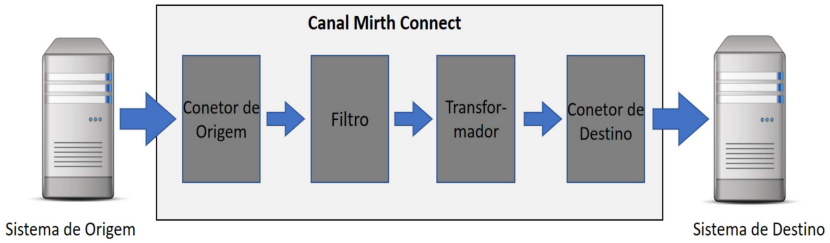


Figura 2.4: Etapas do processamento da informação num canal.

Tal como se pode analisar pela figura 2.4, um canal *Mirth Connect* consiste pelo menos em quatro etapas fundamentais por forma a facilitar o método de processamento da informação recebida.

Conetores são constituintes dos canais que realizam a função de obtenção dos dados e o respetivo envio dos mesmos para determinado(s) sistema(s) externo(s). Desta forma, a informação passa por os seguintes componentes de um canal:

- **Conetor de origem:** tem como principal funcionalidade de receber os dados provenientes do sistema emissor, podendo apenas receber informação de uma só fonte. Através deste conetor existe a opção de configurar filas de espera para as mensagens, captar os dados recebidos e dividir em várias mensagens, escolher quando enviar a resposta de volta ao sistema de origem (*Acknowledgement*) e escolher um número máximo de *threads* de maneira a permitir processar um maior número de mensagens num dado instante.
- **Filtro:** depois das mensagens serem recebidas pelo sistema vão para este componente, onde é determinado se uma mensagem deve prosseguir para a próxima etapa, com base num conjunto de regras definidas pelo utilizador.

- **Transformador:** logo após a seleção das mensagens pretendidas através do filtro, estas são encaminhadas para este componente. Esta fase são realizadas uma série de etapas que podem modificar as mensagens recebidas (converter num padrão diferente, por exemplo) ou extrairem elementos dos dados das respetivas mensagens e mapeá-los em variáveis.

- **Conetor de destino:** tem como principal tarefa o envio dos dados já processados para um ou mais sistemas externos. Por exemplo, pode-se receber dados por *HTTP*, gravá-los numa pasta e inserir partes dos dados numa determinada base de dados. Por intermédio deste conetor permite analisar os sistemas externos ativos para receber informação, possibilita escolher a resposta do sistema externo ao recetor e forçar mensagens de erro para determinadas mensagens.

Esta ferramenta dispõe de várias funcionalidades que auxiliam o utilizador e facilitam o processo de manuseamento da mesma, tais como a configuração de alertas para determinados eventos, monitorização da atividade de mensagens enviadas/recebidas, armazenamento dos *Acknowledgement* por forma a proporcionar uma análise sobre os tempos das mensagens, geração de *logs* do estado das conexões estabelecidas com os sistemas externos, entre outras.

Por forma a permitir a conexão entre vários sistemas, esta plataforma é provida por uma ampla gama de protocolos de transferência. Tal como os protocolos, também possuí uma variedade elevada de formatos e padrões de dados (tabela 2.2) com principal objetivo de alcançar um amplo leque de padrões de maneira a suportar a heterogeneidade de como a informação é apresentada.

Protocolos de Aplicação e de Transferência Suportados	*LLP* *TCP/IP* *HTTP* *Flat Files* *ODBC Databases* *SFTP*	*FTP* *SMB* *WebDAV* *IMAP/POP3* *SMTP* *DICOM*	*JMS* *SOAP* e *REST* (*Web Services*) *Java e JavaScript* *ATSM* E1381
Formato de dados e Padrões Suportados	HL7 v2.x/v3 e FHIR *Delimited Text* (CSV, etc) *DICOM* *EDI*/X12 *XML*		JSON *PDF/RTF Documents* *NCPDP* Raw (qualquer formato de dados) *ATSM* E1394

Tabela 2.2: Protocolos e formatos de dados suportados pela plataforma *Mirth Connect*[8].

2.3.2 *Corepoint Integration Engine*

A organização que desenvolveu este *software* designa-se ***Corepoint Health*** e foi fundada em 1997. Em meados de Julho de 2019, esta companhia uniu-se a outra chamada ***Rhapsody*** com principal objetivo de expandir os seus recursos por forma a atender à crescente necessidade de interoperabilidade entre os diversos provedores e fornecedores de serviços de assistência médica [48].

O *Corepoint Integration Engine* pode ser usado para fornecer a qualquer tipo de unidade de saúde um mecanismo de interface principal, onde se consegue empregar técnicas de encaminhamento, filtragem e transformação de mensagens. O seu suporte ao protocolo HL7 permite o mapeamento das diversas mensagens para uma base de dados.

Esta ferramenta possui uma ampla gama de caraterísticas, as quais estão implícitas dentro de cinco elementos fundamentais [49]:

- **Comunicação:** Permite gerir todas as comunicações com sistemas externos, por forma a possibilitar o fluxo de informação. Como se poderá verificar adiante, pode comunicar através de um grande número de protocolos, dos quais se destacam *TCP*/*IP*, serviços *web* e bases de dados. Através de uma interface *user friendly*, possibilita a criação e configuração de conexões e proporciona a monitorização remota das mesmas. Propicia também a configuração e gestão das confirmações (*ACK*) do envio/receção de mensagens e resolve problemas de implementação e comunicação do *LLP* (garante que aplicações compatíveis com HL7 percebam onde uma mensagem começa/acaba através da adição de *tags*).

- **Mensagens:** Nesta plataforma, todas as mensagens passam por três processos importantes: Encaminhamento, Filtragem e Mapeamento. Na encaminhamento, as mensagens podem ser direcionadas dinamicamente para várias fontes. Na filtragem, as mensagens são encaminhadas para diferentes conexões, conforme o conteúdo. No mapeamento, o conteúdo de interesse do utilizador é mapeado para variáveis, podendo ser utilizado para diferentes fins.

- **Armazenamento:** O registo relativo a cada mensagem é guardado, para cada conexão. Estes *logs* podem ser consultados posteriormente, pelos dias em que foram gerados. No caso de falha de transmissão de uma mensagem, o sistema tem a capacidade de reenviar a mesma pelo reconhecimento da informação que se encontra no seu registo. Como manutenção, esta ferramenta é provida de um mecanismo de limpeza que exclui os *logs* que já possuem um determinado tempo de armazenamento.

- **Configuração e Testes:** Esta plataforma contém mecanismos de teste para verificar a conformidade entre as mensagens recebidas de um sistema externo e o formato de

mensagem esperado. Além disso, fornece suporte para um ambiente de teste para que todo o sistema possa ser testado antes de afetar as mensagens reais de assistência médica.

– **Alertas:** Os alertas são configurados pelo utilizador, permitindo controlo total sobre quais eventos vão acionar um alerta. Estes eventos podem ser orientados às mensagens (falha no envio, mensagem de erro, etc), orientados a conexão (parada, não conectada, inativa, etc), de profundidade de uma fila de mensagens, etc.

Sendo esta uma plataforma de integração é provida de um elevado número de protocolos, tanto a nível de comunicação, como de mensagens [50]:

- HL7 v2.x/v3 e FHIR
- JSON
- *NCPDP*
- X12
- *XML*
- *DICOM*
- *FTP*
- *SOAP* e *REST* (*Web*)
- *LLP*
- *TCP/IP*
- *Microsoft SQL*
- *ODBC Database*

2.3.3 *Iguana*

Este *software* foi desenvolvido pela empresa ***Interfaceware***, em 1997. Esta organização, desde que foi fundada, dedica-se a fornecer meios para permitir o intercâmbio de dados entre diferentes sistemas de saúde.

O *Iguana*, tal como as outras plataformas apresentadas, é um mecanismo de integração HL7 que permite a comunicação entre sistemas díspares e promove um fluxo contínuo de informação. As principais funcionalidades desta ferramenta passa pela análise e gestão de mensagens HL7, filtragem e encaminhamento destas mensagens para sistemas externos, transformação dos dados nos diversos tipos de mensagens HL7 entre outros.

Da mesma forma que o *software Mirth Connect*, a interface entre uma fonte de mensagens e o destino pretendido designa-se de canal. Este tem como principal função direcionar o fluxo de dados de um local para outro. Os dados podem ser enviados e recebidos de diversas fontes [51]:

- *LLP Listener/Client*: Fica à escuta numa determinada porta *TCP*/*IP* no endereço pretendido. Este protocolo é o mecanismo de transporte HL7 não criptografado mais comum através de uma rede local.
- Bases de Dados: Especifica que os dados podem provir ou ser guardados numa base de dados. Para tal esta ferramenta recorre a uma interface chamada *Chameleon*, que permite o mapeamento e transformação dos dados.
- *Plugins*: Possibilita a conexão a aplicações externas para envio de mensagens, onde o *Iguana* pode configurar e controlar o fluxo.
- Ficheiros: Permite guardar a informação, numa máquina local, em ficheiros (*XML*, *JSON*, etc).
- Filtros: Regras definidas pelo utilizador, que encaminha as mensagens para determinados canais, conforme o conteúdo.
- Canal: O destino pode ser pré definido para um ou mais canais do *software*.
- *HTTPS*: Tanto as mensagens recebidas como as enviadas podem ser procedentes de aplicações cliente/servidor.

Esta plataforma permite ainda a pré definição de alertas, registo de atividade das mensagens, monitorização das métricas em tempo real, criptografia dos dados que circulam, ferramenta de implantação (usada para simplificar a promoção de interfaces durante todo o processo de desenvolvimento, teste e produção), *logs* com o histórico de atividades do utilizador e muitas outras utilidades.

Tal como se pôde observar anteriormente, esta ferramenta tem a possibilidade de estabelecer conexões com uma ampla diversidade de sistemas externos. Neste âmbito, também apresenta suporte para vários protocolos de comunicação, bem como pode processar tipos distintos de dados (tabela 2.3).

Protocolos de Aplicação e de Transferência Suportados	*LLP* *TCP/IP* *HTTPS* *ODBC Databases* *SFTP*	*FTP* *DICOM* *SOAP* e *REST* (*Web Services*) *File System*
Formato de dados e Padrões Suportados	HL7 v2.x/v3 e FHIR *Delimited Text* (CSV, etc) *DICOM*	*EDI*/X12 *XML* JSON

Tabela 2.3: Protocolos e formatos de dados suportados pela plataforma *Iguana* [9].

2.3.4 *Comparação entre os softwares*

De forma a selecionar um motor de integração para analisar durante todo o trabalho desta dissertação, foi realizada uma pesquisa acerca de soluções que permitissem a integração do *standard* de comunicação de mensagens HL7, por ser um dos mais utilizados por sistemas no âmbito de unidades de saúde.

A solução escolhida deve ter caraterísticas que sejam comuns à maioria das plataformas HL7, ou seja, permitir a receção, a validação da informação, realização de transformações com base em regras lógicas e o envio de mensagens em diversos formatos, incluindo o formato HL7.

Na tabela 2.4, é apresentada uma comparação entre as soluções descritas anteriormente e outras soluções emergentes. O *software Corepoint Integration Engine* foi apresentado anteriormente por ser consecutivamente classificado pela organização **KLAS *Research*** um dos melhores sistema de integração e à grande quantidade de informação referente às suas caraterísticas técnicas, no entanto, não se encontra na comparação pois não foi possível a obtenção de uma versão de teste para proceder à sua análise.

		Mirth Connect	Iguana	eiConsole	HL7 Soup
Standards suportados	**HL7 v2.x**	Sim	Sim	Sim	Sim
	HL7 v3	Sim	Sim	Sim	Sim
	XML e CSV	Sim	Sim	Sim	Sim
	DICOM	Sim	Não	Sim	Não
	Outros	NCPDP, X12 EDI, FHIR e JSON	X12 EDI, JSON e FHIR	X12 EDI, JSON e FHIR	FHIR e JSON
Bases de dados suportadas	**MySQL**	Sim	Sim	Sim	Sim
	Oracle	Sim	Sim	Sim	Sim
	PostgreSQL	Sim	Sim	Sim	Sim
	SQL Server	Sim	Sim	Sim	Sim
	Outras	Derby [1]	MS Access, DB2, Firebird, Sybase	Derby, MongoDB	OleDB
Protocolos de Comunicação Suportados	**TCP/IP**	Sim	Sim	Sim	Sim
	MLLP	Sim	Sim	Sim	Sim
	HTTP	Sim	Sim	Sim	Sim
	FTP	Sim	Sim	Sim	Não
	Web Services	Sim	Sim	Sim	Sim
	Outros	RTF e JMS	Não	JMS	Não
Especificações internas	**Transformação das mensagens**	Sim	Sim	Sim	Sim
	Validação de mensagens	Sim	Não	Sim	Sim
	Mapeamento de mensagens	Sim	Sim	Sim	Sim
	Filtro de Mensagens	Sim	Sim	Sim	Sim
	Interface para monitorização de mensagens	Sim	Sim	Sim	Sim
	Linguagem de programação	JavaScript	Lua	Java	C#
Sistemas Operativos suportados	**Windows**	Sim	Sim	Sim	Sim
	Linux	Sim	Sim	Sim	Não
	Outros	Mac OS X	Mac OS X	Mac OS X, IBM AIX, HP/UX	Não
Modelos de Implementação	**Docker**	Sim	Não	Sim	Não
Outros	**Licença**	*Open Source* Comercial	Comercial	Comercial	Comercial

Tabela 2.4: Comparação entre diferentes *softwares* de integração.

1 Através de *scripts* internos em JavaScript podem ser utilizadas outras bases de dados, como do tipo não relacionais.

2.3.5 *Conclusão*

De acordo com o trabalho a ser desenvolvido, a escolha da ferramenta de integração terá de atender aos seguintes requisitos:

- Compatibilidade com os principais *standards* utilizados para partilha de informação na área da saúde;
- Suporte para diferentes tipos de conetores, de modo a compreender uma maior variedade de sistemas de integração a testar;
- *Open Source*, ou seja, um *software* livre sem custos de licenciamento;
- Suporte às funcionalidades comuns de uma plataforma de integração: validação, filtragem, transformação e encaminhamento de mensagens;
- Interface de monitorização inerente ao fluxo de mensagens;

Com base nestes requisitos e na tabela 2.4, a plataforma *Mirth Connect* destaca-se por ser um sistema de *software* robusto e abrangente. Mesmo sendo o único sistema de integração *open source*, possui as especificações necessárias para igualar ou superar as ferramentas concorrentes. Uma caraterística importante é o facto de fornecer ao utilizador o desenvolvimento de *scripts*, de forma a criar condições para atender às suas necessidades, caso estas não se encontrem nas funcionalidades da interface. Deste modo, permite a conexão a diferentes tipos de fontes não incluídas no *software*, como bases de dados *noSQL*. Outra grande vantagem relaciona-se com o facto de poder ser implementada através do *Docker*, tornando possível a criação de ambientes de teste.

2.4 MONITORIZAÇÃO

A palavra monitorização pode surgir em vários contextos e de diferentes maneiras, no entanto, apresenta um conceito que é unívoco a todas as vertentes. A monitorização, em termos gerais, consiste no ato de "supervisionar, acompanhar e avaliar", "controlar mediante acompanhamento" ou "olhar atentamente ou controlar com um propósito específico" [52]. Refere-se a um processo constante que se carateriza essencialmente por três componentes importantes: a recolha de dados, a análise periódica dos dados e a disseminação das informações adequadamente analisadas a todas as entidades intervenientes [53].

Relativamente aos sistemas computacionais no domínio da saúde, a monitorização abrange a análise a avaliação da performance dos diversos serviços com o auxílio de sistemas de comunicação e de computadores. Para proceder corretamente com a monitorização de um determinado sistema, inicialmente é necessário conhecer as suas adversidades, ou seja, é

preciso definir objetivos por forma a alcançá-los. Para isto, é indispensável dimensionar a carga a que se encontram submetidos e eleger as melhores métricas, tendo em conta os objetivos estabelecidos.

Uma característica essencial de um componente é que seu desempenho depende bastante da carga de trabalho a que está sujeito, sendo determinante o seu conhecimento. Considerando, por exemplo, a avaliação do desempenho de um servidor *web*. Poder-se-ia qualificar a carga como o número de solicitações por segundo (intensidade da carga de trabalho). No entanto, o desempenho de um sistema não depende apenas da intensidade da carga de trabalho, mas também de sua natureza. Por exemplo, no servidor *web* nem todas as solicitações são equivalentes, ou seja, um pedido para o método *get* pode ter um bom desempenho para objetos frequentemente usados e em contrapartida, o mesmo servidor *web* pode ter um fraco desempenho ao aceder às bases de dados. Noutros servidores *web*, a situação pode-se alterar ou ser outra. Desta forma, é necessário ter consciência das tarefas de cada componente, por forma a proceder a uma avaliação apropriada [52].

Em relação à seleção das métricas a serem consideradas para o processo de monitorização, requer um elevado conhecimento acerca do sistema. Através da definição das métricas e dos indicadores da carga de trabalho de um determinado componente, são encontrados todos os parâmetros aos quais é necessário recolher informação [52]. Pela monitorização eficiente, é possível caracterizar um sistema e reconhecer as suas vulnerabilidades e ameaças. Para além disto, os administradores do sistema conseguem implementar sistemas de apoio à decisão para efeitos de balanceamento de recursos [53].

2.4.1 *Medidas de Desempenho*

Como especificado anteriormente, o processo de monitorização passa por determinar a carga de determinado sistema e a seleção de indicadores para uma posterior análise e avaliação do desempenho. Desta forma, deve-se ter em consideração a carga a que os sistemas estão submetidos e ainda as métricas necessárias para atingir os objetivos estipulados.

Pretende-se a monitorização das máquinas de uma plataforma de integração hospitalar, tendo em vista a sua melhor performance. Para tal, torna-se necessário ter em vista alguns parâmetros, caraterísticos da máquina, que intervêm na carga de trabalho e que tem influência direta no modo como o *software* atua:

- **Percentagem de *CPU* livre:** este parâmetro significa a percentagem de tempo que o processador gasta ao executar a *thread* do System Idle Process (processo que é executado pelo sistema quando nenhum outro processo necessita de utilizar o *CPU*). *Central Process Unit (CPU)*, ou unidade de processamento central, é pensado como se fosse o cérebro de um computador. É responsável pelo processamento de todas as operações do computador: operações aritméticas básicas, lógicas, de controlo e de

input/output especificadas por instruções. Desta forma, pode-se afirmar que as suas características influenciam diretamente na velocidade com que as suas aplicações vão executar numa dada máquina [54].

- **Percentagem de memória livre:** percentagem da memória física disponível para os processos que se encontram em execução. A memória livre é resultado da soma de três conceitos: lista de modo de suspensão (memória recentemente removida de um processo terminado), lista livre (memória pronta a ser utilizada) e lista zero (retrata as páginas de memória preenchidas a zeros, para prevenir que processos posteriores tenham acesso a processos anteriores). Esta memória é relativa à *RAM* (*Random Access Memory*), que recebe as informações do disco e armazena-as temporariamente, disponibilizando-as ao processador [55].

- **Percentagem de disco livre:** esta medida é a relação entre o espaço disponível e o espaço total que pode ser utilizado por todas as unidades de disco lógico instaladas em determinada máquina [56]. O disco rígido, também designado como disco duro, é um componente de hardware do computador onde os dados são armazenados. Ao contrário da memória *RAM*, é considerado uma memória não volátil, isto é, quando a máquina é desligada as informações não são perdidas [56].

2.5 TRABALHO RELACIONADO

Como já foi referido, o desenvolvimento desta dissertação passa por a análise e otimização da plataforma de interoperabilidade *Mirth Connect*, com base em indicadores de desempenho previamente recolhidos. No entanto, no que diz respeito a trabalhos relacionados com a análise de plataformas de integração na área da saúde existe muito pouca informação.

Para colmatar esta escassez relativamente a trabalhos de investigação nesta vertente, foram consideradas abordagens que promovem o fluxo contínuo de dados e trabalhos que referenciam indicadores de análise que permitem uma monitorização fiável. Neste contexto, esta pesquisa foi realizada para *softwares* que integrem uma infraestrutura coincidente com a do objeto de estudo desta dissertação e da maioria das ferramentas de integração do mesmo domínio, ou seja, soluções *Message Oriented Middleware* (*MOM*).

2.5.1 *Filas de Mensagens*

Hoje em dia frequentemente os sistemas de *software* são distribuídos, e sendo estes maioritariamente executados em dispositivos heterogéneos, estabelecer a conexão entre eles de forma simples e confiável pode constituir um desafio.

Uma solução estabelecida para esta adversidade, designa-se de *Message Oriented Middleware* (*MOM*). Também desenvolvido em *Java* pelo *Java Message Service API*, tem como objetivo fundamental estabelecer a troca de mensagens de forma ágil, inviabilizando a sincronização, coordenação e comunicação entre determinados sistemas distribuídos.

Esta abordagem geralmente é provida de dois tipos diferentes de comunicação (figura 2.5) [57].

- ***Point-to-point***: também conhecido por *Queue Model*, constituído por dois intervenientes principais: um remetente e um destinatário. O remetente envia a mensagem para a fila e apenas um destinatário a recebe. Aquando a confirmação da receção da mensagem, esta é excluída da fila. Deste modo, sendo que cada mensagem é enviada apenas a um destinatário, este modo de entrega é adequado para o trabalho de balanceamento de carga entre consumidores.
- ***Publish/Subscribe***: contrariamente ao modelo anterior, neste tipo de comunicação existe um remetente e um ou mais destinatários, que são chamados de *publishers* e *subscribers*, respetivamente. O *publisher* envia a mensagem para determinado tópico, e uma cópia da mensagem é encaminhada para todos os *subscribers* que se inscreveram a esse tópico. A mensagem permanece no tópico até que todos os destinatários a recebam ou até que a mensagem expire.

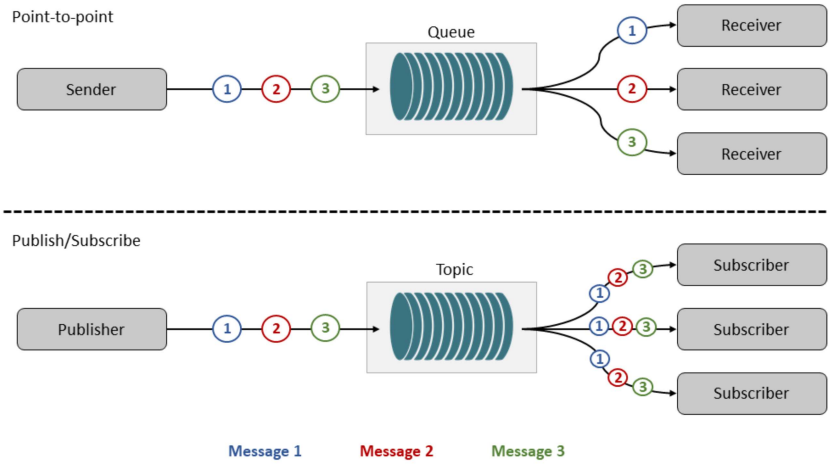


Figura 2.5: Diferença entre os modelos *Point-to-point* e *Publisher/Subscriber* (adaptado de [4]).

As ferramentas de integração geralmente utilizam a noção de filas de mensagens em que pode ocorrer a possibilidade de existirem diferentes destinatários para apenas um emissor.

Visto isto, pelo que foi referenciado anteriormente, uma fila de mensagens é uma técnica utilizada para a comunicação entre vários sistemas, através de um protocolo definido ou de uma interface para permitir a passagem de mensagens.

Esta técnica pode adicionar alguma complexidade relativamente ao design de determinado sistema de *software*, no entanto, principalmente para aplicações que realizem processamento de fluxo de mensagens é uma mais valia, para gerir a complexidade inerente a uma escala de dados adicional.

As principais caraterísticas que a implementação de uma fila de mensagens deve providenciar são [58]:

- **Armazenamento**: Geralmente armazena as mensagens em algum tipo de *buffer* até que estas sejam lidas por um processo de destino ou explicitamente removidas da fila de mensagens.
- **Assincronismo**: O armazenamento num *buffer* de mensagens permite que as filas exponham um grau de assincronicidade quanto aos sistemas envolvidos, permitindo que os processos de origem enviem mensagens e as deixem acumular numa fila, enquanto que os processos de destino vão selecionando-as para processamento. Este mecanismo permite que as aplicações funcionem em determinados cenários de falha, como conectividade intermitente ou falha do processo de origem ou de destino.
- **Encaminhamento**: As filas de mensagens podem também fornecer funcionalidades de encaminhamento para as mensagens, onde vários processos de destino podem ler ou gravar mensagens da mesma fila.

Esta implementação traz inúmeras vantagens quanto à sua utilização, das quais se destacam [58]:

- **Evita conexões diretas**: Proporciona a comunicação indireta entre sistemas, ou seja, um sistema A não precisa de conhecer um sistema B para se sincronizarem e estabelecerem uma ligação. Basta os sistemas estarem conetados a um gestor de filas, para o sistema A colocar mensagens na fila e para o sistema B adquiri-las.
- **Comunicação independente do tempo**: As mensagens permanecem num buffer até que sejam removidas por um programa destino. Isto significa que o programa que envia as mensagens pode continuar com o processamento sem esperar que o destinatário confirme o recebimento de uma mensagem (comunicação assíncrona), assim como o programa de destino não precisa de estar em execução quando dada mensagem é enviada.
- **Comunicação pode ser orientada a eventos**: os sistemas que enviam/recebem mensagens podem ser controlados a partir do estado das filas. Por exemplo, o inicio

da execução de determinado programa pode ser programado apenas quando uma mensagem chegue à fila.

- **Prioridade de mensagens**: Pode ser atribuída uma prioridade a uma mensagem quando é colocada numa fila. Isso determina a posição na fila em que a nova mensagem é adicionada. Desta forma, as mensagens podem ser obtidas na ordem em que as mensagens estão na fila ou pela aquisição de uma mensagem específica.
- **Suporte de recuperação**: geralmente os mecanismos de filas de mensagens possibilitam persistência e registo, permitindo a recuperação do estado e das mensagens na fila durante falhas. Isto pode ser feito através de um backup realizado em bases de dados ou outro tipo de armazenamento provisório.
- **Facilidade quanto à escalabilidade**: as filas podem ser escalonadas horizontalmente para lidar com um aumento na carga de mensagens, ou seja, a carga pode ser balanceada através da sua divisão por outros sistemas.

2.5.2 *Processamento de fluxo de dados*

Como já referenciado anteriormente, a problemática implícita ao trabalho desta dissertação relaciona-se com o envio e receção de grandes volumes de dados em plataformas de integração, que muitas das vezes, reflete uma sobrecarga neste tipo de *software*. Neste contexto, torna-se essencial abordar soluções que promovam o processamento do fluxo de dados, através do estudo de caraterísticas que um sistema deve possuir, de forma a inviabilizar a passagem e processamento de grandes quantidades de informação.

Uma das temáticas que enfrenta este tipo de desafios é a noção *Big Data*, onde uma das suas maiores adversidades relaciona-se com a sua crescente escalabilidade, em que vários sistemas emissores enviam continuamente grandes conjuntos de dados distribuídos, que são processados por várias horas de forma a produzir grandes quantidades de informação. O conceito desta área de conhecimento, é constituído por 5 noções principais que também fazem parte dos objetivos intrínsecos a plataformas de integração [59]:

- **Volume**, que representa a grande quantidade de dados.
- **Variedade**, a diversidade de informação que é recebida/transmitida.
- **Veracidade**, toda a informação incluída num fluxo deverá ser verdadeira;
- **Valor**, referenciando a informação útil propiciada.
- **Velocidade**, em que todo o processamento deve ser ágil para gerar informações úteis.

O paradigma de processamento de fluxo aplica operações a cada elemento de dados, emitido por uma fonte de dados de entrada infinitamente longa. O objetivo deste conceito é amenizar impactos negativos oriundos da crescente escala de dados provenientes de diversas fontes. Em seguida são descritas oito regras que um sistema deve possuir de maneira a promover um bom desempenho quanto ao processamento de fluxo em tempo real [60].

1. Manter os dados em movimento

Para evitar a introdução de latências quanto ao fluxo dos dados, o sistema deverá ser capaz de realizar o processamento de mensagens *in-stream* sem ter associadas operações de armazenamento, ou seja, sem ter necessidade de armazená-las no disco ou em bases de dados durante o seu caminho crítico. Apesar disto, os sistemas devem ser ativos (orientados a eventos) e não passivos, onde existem atrasos relacionados com pesquisas de resultados para detetar condições de interesse.

2. Suporte a consultas usando uma linguagem *SQL*

SQL surgiu como uma linguagem tradicional para consulta de dados, no entanto, esta linguagem opera numa quantidade fixa de dados, em que quando uma consulta chega ao final da tabela informa que esta foi concluída. Em cenários de *streaming*, como os dados têm tendência a aumenta continuamente, foi estabelecida uma linguagem *StreamSQL*. Esta possibilita a criação de várias janelas deslizantes que são definidas conforme parâmetros, como número de mensagens e instâncias de tempo variáveis.

3. Lidar com falhas de fluxo

Em sistemas de tempo real, os dados podem ser perdidos, chegar atrasados ou fora de ordem. Um sistema de processamento de fluxo não pode esperar indefinidamente pelos dados, mas também não pode ignorar ou perder nenhum dado. Sendo assim, estes sistemas devem ser resilientes contra imperfeições de fluxo, providos de mecanismos pelo qual a chegada de uma mensagem tardia possa ser aceite.

4. Gestão de resultados previsíveis

O resultado de qualquer sistema de processamento de fluxo deve ser determinístico e reproduzível para o mesmo fluxo. As mensagens devem ser produzidas em ordem de tempo crescente, independentemente da hora de chegada.

5. Integração do estado dos dados

As aplicações de processamento de fluxo geralmente devem combinar o presente com o passado. O armazenamento dos dados deve ser realizado (através de uma base de dados, por exemplo), para permitir o uso de uma linguagem uniforme que lida com dados armazenados e de streaming. Com este processo, através de uma pesquisa é fornecido o histórico de determinada mensagem.

6. Garantir alta disponibilidade

Normalmente os sistemas de processamento de fluxo em tempo real não suportam recuperações de reinicialização. De forma a evitar perda de dados, os sistemas devem permitir constantemente a transição rápida de dados para um outro sistema secundário de *backup*, estando regularmente sincronizado com o primário.

7. Escalonamento e balanceamento automático

Uma arquitetura de processamento de fluxo ideal deve ser sem quebras ou bloqueios, explorando arquiteturas *multithread*. Além disso, deve ser capaz de lidar com o dimensionamento do sistema de forma automática, adicionando ou removendo máquinas, com base em crescentes ou decrescentes volumes de dados, ou até pela utilização intensiva dos recursos de determinada máquina.

8. Avaliação

Todos os componentes do sistema devem ser pensados de forma a obter um alto desempenho, tendo em atenção a minimização da proporção de sobrecarga para trabalho útil. O sistema deve ser testado e avaliado com base na sua carga de trabalho, onde as metas de taxa de transferência e latência devem ser validadas.

2.5.3 *Métricas de Desempenho*

Tendo em conta a arquitetura da ferramenta de integração utilizada, neste sub-capítulo serão descritos trabalhos de investigação relacionados, que realizem a avaliação de performance de sistemas de *software* que integrem mecanismos de comunicação *MOM*. Visto isto, dos trabalhos apresentados serão evidenciadas as métricas que os autores utilizam para proceder à monitorização do desempenho dos *softwares*.

Chen et al., num artigo intitulado de "*QoS Evaluation of JMS: an Empirical Approach*", estabeleceu a relação entre *Java Message Service* e *Message Oriented Middleware* [61]. De modo a tornar claro, *MOM* é uma infraestrutura de suporte à comunicação entre componentes de

software e aplicações, enquanto que *JMS* é a implementação desse modelo em *Java*. O tipo de comunicação utilizada na respetiva comunicação foi *point-to-point*. Desta forma, os autores apresentaram diferentes métricas a fim de avaliar o desempenho destas duas abordagens baseadas em mensagens:

- ***Maximum Sustainable Throughput***: É o nível onde a diferença entre a taxa de envio de mensagens e a taxa de receção é zero, calculando o seu rendimento. A partir deste nível diferentes taxas de mensagens são verificadas. No caso desta diferença representar um valor maior que zero, significa que a taxa de envio é superior à de receção. Sendo assim, uma taxa acima de uma diferença nula, pode representar uma acumulação de mensagens não sustentável no servidor, acabando por utilizar indevidamente recursos (como consumo de memória).
- **Latência**: Define o tempo gasto para a entrega de mensagens do sistema fonte ao sistema recetor. É medida a partir do momento em que a mensagem é publicada/enviada para a fila até o momento em que a mensagem chega ao subscritor/recetor.
- **Envio de conjuntos de mensagens**: Através deste indicador é realizada a avaliação do tempo total gasto para controlar o envio e receção de um determinado conjunto de mensagens.
- **Mensagens persistentes após a recuperação**: As mensagens persistentes são colocadas em filas, com propósito de não serem perdidas quando ocorrem falhas no servidor. Para tal, foram calculadas o número de mensagens recebidas e, após uma falha do servidor, é feito o cálculo das mensagens enviadas de forma a obter o número de mensagens recuperadas, com objetivo de verificar se existiram perdas de mensagens.

Num outro trabalho de investigação, foi analisada a eficácia entre duas ferramentas que usam infraestruturas *MOM* na sua implementação: *Tibco Rendezvous* (TIB / RV) e *Progress SonicMQ* [62]. Este artigo de título *"Benchmarking Message-Oriented Middleware – TIB/RV vs. Sonic MQ"* foi escrito por M. Pang et al., e fornece uma boa base para uma comparação funcional entre os *middlewares* em estudo. Estes *softwares* podem ter os modos de entrega de mensagens *point-to-point* ou *publish/subscribe*, sendo avaliadas estas duas vertentes durante o documento. Com base nas caraterísticas destes dois *softwares*, os indicadores tecidos para avaliação de performance foram as mensagens enviadas/recebidas por segundo, o tempo que demora um dado conjunto de mensagens a ser enviado, os efeitos causados por o aumento do número de *publishers*/remetentes e *subscribers*/recetores, a comparação de um envio constante de mensagens em oposição por envios definidos em determinados períodos de tempo e a utilização dos recursos da máquina onde são realizados os testes (consumo de CPU e memória).

P. Dobbelaere et al., no artigo *"Industry Paper: Kafka versus RabbitMQ"*, é outra referência usada nesta investigação que compara o desempenho entre dois *Message Oriented Middlewares (MOMs)*, do tipo *publish/subscribe* [63]. Esta comparação é realizada através dos indicadores de performance de *throughput* e latência de mensagens, tendo em conta diferentes configurações de garantia de entrega de mensagens. Estas garantias de entrega, normalmente são constituídas por 3 variantes:

- **No máximo uma vez:** Não existe garantia que todas as mensagens cheguem ao consumidor, pelo facto de poderem ser perdidas durante falhas. No entanto, garante que um consumidor não receba uma mensagem mais que uma vez.
- **Pelo menos uma vez:** Todas as mensagens têm garantia de entrega ao consumidor, pelo intermédio de duas etapas, a unidade de gestão de mensagens do *middleware* (*broker*) reconhece todas as mensagens que recebe do produtor e o consumidor reconhece todas as mensagens que recebe do *broker*. Com isto, é possível que o consumidor receba mensagens duas vezes quando uma confirmação se perde durante as falhas.
- **Exatamente uma vez:** Todas as mensagens são entregues ao consumidor. Requer um processo dispendioso, com um tipo de comunicação *point-to-point*, onde o produtor reconhece o consumidor e acompanha todo caminho da mensagem.

Como se pode constatar, da pesquisa da literatura fornecida, existem estudos que discutiram e quantificaram o desempenho de Message Oriented Middleware comerciais. Neste contexto, esta pesquisa bibliográfica não revelou nenhum estudo pertencente a *softwares* de integração que incluíssem mecanismos MOM para fins no domínio da saúde, no entanto, os documentos abordados foram proveitosos para fornecer e sugerir métricas para o trabalho desta dissertação.

3

METODOLOGIA DE INVESTIGAÇÃO

Cada caso de estudo apresenta uma metodologia de investigação característica, consoante os objetivos a serem atingidos. Por a forma a facilitar esta organização que leva ao desenvolvimento de diferentes tipos de trabalhos, existem várias abordagens já definidas que são adequadas a diversos tipos de investigação pretendidos.

Por conseguinte, neste capítulo optou-se por abordar uma metodologia designada *Design Science Research* que serve de apoio a toda a implementação inerente ao trabalho desenvolvido nesta dissertação.

3.1 DESIGN SCIENCE RESEARCH

A metodologia de investigação *Design Science Research (DSR)* faz parte da pesquisa na área das Tecnologias de Informação desde a década de 1990, sendo um conceito imprescindível pois permite projetar sistemas de trabalho com base num modelo definido, melhorando o seu desempenho organizacional. Tem como principal objetivo a construção e avaliação de objetos, modelos ou métodos designados artefactos, que permitem aos profissionais resolver problemas, com base no processamento de informação organizacional e desenvolvimento de ações [64, 5]. Este modelo é útil, quer para a criação de um novo artefacto, quer para a melhoria de sistemas já existentes, defendendo que o conhecimento advém da construção.

Neste contexto, esta abordagem consiste num processo rigoroso para projetar e desenvolver artefactos bem sucedidos, conforme problemas observados. Estes artefactos representam um objeto de estudo que se pretende desenvolver, ou por outras palavras, o problema a solucionar. Podem variar entre aplicações de *software*, lógica formal, equações matemáticas complexas, entre outros [65].

Assim sendo, torna-se essencial que o artefacto projetado corresponda a uma solução tecnológica viável para a resolução de problemas de negócio importantes, de modo a que a sua utilidade, qualidade e eficácia seja demonstrada através de métodos de avaliação bem executados. Além disso, a pesquisa científica realizada, deve fornecer contribuições claras e deve basear-se na aplicação de métodos no seu processo de construção e avaliação [64, 66].

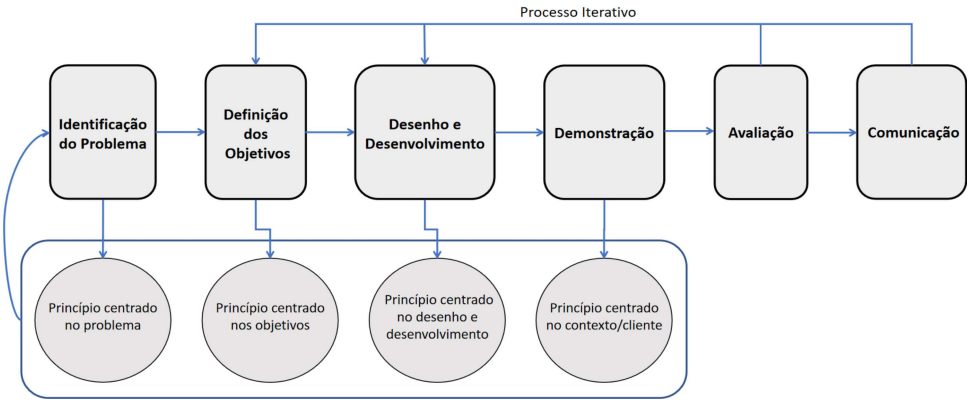


Figura 3.1: Diversas fases do processo da *Design Science Research* (adaptado de [5]).

Na figura 3.1 está representado o modelo que retrata a metodologia DSR, apresentado por *Ken Peffers* [5]. De seguida, é apresentada uma breve descrição relativa a cada um dos passos representativos deste modelo [5, 67]:

a) **Identificação do Problema:** Definir o problema específico de investigação e justificar o valor da solução. É necessário que seja entendido a relevância do artefacto a desenvolver e identificar as suas soluções atuais. Deste modo, o artefacto em questão deve ser algo que demonstre uma clara contribuição ao ambiente de negócio em que se insere, resolvendo uma adversidade anteriormente não ultrapassada.

b) **Definição dos Objetivos:** Compreender os objetivos necessários para a solução, através da definição do problema e conhecimento do que é possível e viável. Os objetivos a atingir podem ser quantitativos, isto é, termos em que uma solução desejável seria melhor do que as existentes, ou qualitativos, por exemplo, uma descrição de como se espera que um novo artefacto suporte soluções para problemas ainda não abordados. A partir da especificação do problema, os objetivos podem ser inferidos com conhecimento prévio do estado dos problemas, das soluções atuais, e caso existam, e sua eficácia.

c) **Design e Desenvolvimento:** Etapa onde se procede á criação do artefacto. Esses artefactos são potencialmente construções, modelos e métodos. Pretende-se que esta atividade inclua determinar a funcionalidade desejada do artefacto e sua arquitetura e, em seguida, criar o artefato real. Os recursos necessários, passando do passo anterior para o atual, incluem o conhecimento da teoria que pode ser aplicada numa solução.

d) **Demonstração:** Demonstrar o uso do artefacto para resolver uma ou mais instâncias do problema. Isso pode envolver seu uso através de casos de teste, simulação ou uma outra atividade apropriada. Os recursos necessários para a demonstração incluem conhecimento eficaz de como usar o artefacto para resolver o problema.

e) **Avaliação:** Esta fase envolve a comparação de uma solução com os resultados reais observados do uso do artefacto na demonstração. Requer o conhecimento de métricas e técnicas de análise apropriadas. Dependendo de onde provém o problema e o artefacto, a avaliação pode incluir itens como uma comparação da funcionalidade do artefacto com os objetivos da solução estabelecidos. Concetualmente, essa avaliação deve incluir qualquer evidência empírica propícia ou prova lógica. No final deste processo, o investigador pode optar por retornar à etapa (c) por forma a tentar melhorar o artefacto criado, ou pode decidir prosseguir para a fase seguinte e deixar como sugestão algumas melhorias adicionais.

f) **Comunicação:** Etapa final onde se divulga o problema, utilidade e a sua importância, tal como o design e eficiência do artefacto a investigadores e a outro público abrangente, como profissionais, quando apropriado. Desta forma, aquando as publicações de investigações académicas, os investigadores podem usufruir da utilização da estrutura desse processo para estruturar o artigo. Por exemplo, a estrutura nominal de um processo de investigação empírica (definição de problemas, revisão da literatura, desenvolvimento de hipóteses, recolha de dados, análise, resultados, discussão e conclusão) representa uma estrutura comum para trabalhos empíricos de investigação.

Apesar de este modelo se encontrar estruturado numa ordem nominal sequencial, os investigadores podem começar em qualquer etapa e seguir em frente. Uma abordagem centrada no problema é a base da sequência nominal (a), onde a sequência se aplica a casos que resultam da observação do problema ou da investigação futura sugerida num artigo de um projeto anterior. Uma solução centrada nos objetivos (b), pode suscitar num ambiente industrial ou na necessidade de investigação que deve ser abordada através do desenvolvimento de um artefacto. Uma abordagem centrada no desenho e no desenvolvimento (c), resulta na existência de um artefacto que ainda não tenha sido pensado como uma solução, isto é, um artefacto que poderia vir de outro domínio da investigação, podendo já ter sido utilizado para resolução de um problema diferente. Por fim, uma solução centrada no cliente/contexto (d), baseia-se na observação de uma solução prática que já funciona [67, 9].

3.2 COMBINAÇÃO DA METODOLOGIA

É com base no estudo e no resultado final de todas as diferentes fases metodológicas anteriormente descritas, que foi desenvolvida a presente dissertação. De uma forma organizada,

a tabela 3.1 representa a aplicabilidade da abordagem metodológica aos vários capítulos existentes.

Capítulos\Fases	Identificação do Problema	Definição dos Objetivos	Design e Desenvolvimento	Demonstração	Avaliação	Comunicação
1- Introdução	X	X				
2- Estado da Arte			X			
4- Plataforma de Integração			X			
5- Tecnologias de Desenvolvimento			X			
6- Arquitetura e Implementação				X	X	X
7- Conclusão e Trabalho Futuro						X

Tabela 3.1: Fases da Metodologia aplicadas aos capítulos da dissertação.

As duas primeiras fases, Identificação do Problema e Definição dos Objetivos, estão presentes no capítulo 1, onde são introduzidas todas as questões relacionadas com esta dissertação, tal como a contextualização, a explicação da necessidade deste trabalho e os objetivos precisos para o desenvolver.

De seguida, a fase do Design e Desenvolvimento, aplica-se aos capítulo 2, 4 e 5, que procura definir conceitos inerentes a plataformas de integração e definições importantes que serão utilizadas numa fase posterior, o estudo da plataforma de integração a ser analisada e a descrição de tecnologias que darão suporte ao desenvolvimento da arquitetura a ser usada.

As fases demonstração e avaliação são colocadas em prática no capítulo 6, onde é dado o início ao artefacto para melhorar o desempenho do motor de integração, ou seja, são apresentadas métricas de desempenho que podem constituir uma estrutura lógica ideal em determinado ambiente.

Finalmente, a fase da Comunicação será abordada nos capítulos 6 e 7, onde é realizada uma breve análise dos resultados obtidos, dificuldades apresentadas ao longo do desenvolvimento do trabalho e algumas sugestões para trabalho futuro.

4

PLATAFORMA DE INTEGRAÇÃO - MIRTH CONNECT

De acordo com o que já foi referenciado anteriormente, a plataforma de integração a ser analisada nesta dissertação designa-se *Mirth Connect*. Sendo este um *middleware* que recebe, processa e transmite mensagens, torna-se essencial estudar o tipo de fontes e recetores que suporta, bem como a componente funcional, no que diz respeito ao modo de funcionamento dos canais. Para tal, através da análise da plataforma e respetivo estudo com o auxílio da documentação disponível, foi gerado um conhecimento significativo acerca do funcionamento desta plataforma.

Visto o objeto de análise deste trabalho, este capítulo é destinado ao estudo detalhado acerca das noções inerentes a este *software*, com objetivo de proporcionar um conhecimento sólido acerca do seu funcionamento.

4.1 ARQUITETURA DO MIDDLEWARE

De forma a atender a todos os desafios abordados na contextualização, esta ferramenta foi projetada com base numa arquitetura híbrida, estilo cliente-servidor e arquitetura *Enterprise Service Bus (ESB)*, oferecendo suporte ao desenvolvimento de interfaces para mover dados entre dois ou mais sistemas. Deste modo, primeiramente torna-se essencial ter uma noção acerca destas duas arquiteturas:

- **Cliente-Servidor**: arquitetura de *software* em que os clientes enviam sempre pedidos aos quais o servidor responde conforme as solicitações, permitindo uma comunicação entre estes dois intervenientes, onde cada um desempenha diferentes funções [68]. É geralmente composta por um servidor de aplicações (que contém toda a lógica), um servidor de bases de dados e computador (cliente). A comunicação entre os dois componentes, geralmente é realizada em rede (por exemplo, via *TCP* ou *HTTP*).
- **Enterprise Service Bus**: infraestrutura de integração baseada em mensagens (MOM), que fornece serviços de *routing*, invocação e de mediação, para facilitar as interações de sistemas externos de maneira segura e confiável. Este processo é geralmente realizado por meio de *containers* distribuídos, onde estão hospedados serviços de integração

como encaminhadores, transformadores e adaptadores de aplicações, fornecendo-lhes uma ampla variedade de recursos de comunicação. Normalmente, as soluções usando esta arquitetura são construídas sob sistemas de middleware *JMS*, ou seja, que garantem a receção e entrega de mensagens (como o *Mirth*). Uma solução *open source* para a implementação desta infraestrutura designa-se de *Mule* [69].

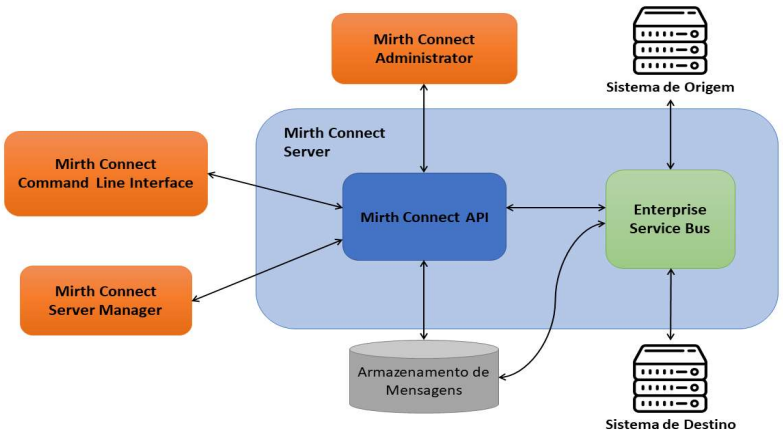


Figura 4.1: Componentes constituintes da arquitetura do *Mirth Connect*.

Depois de se entender as arquiteturas abordadas, torna-se mais simples a compreensão de como esta plataforma está organizada. Como se pode observar pela figura 4.1, o *Mirth* é constituído por 4 componentes essenciais:

- ***Mirth Connect Server*:** contém o *back-end* necessário para a interface de gestão e o mecanismo de integração (filtro, transformador, conetores). É neste componente onde está implícita toda a lógica deste sistema de *software*.
- ***Mirth Connect Administrator*:** interface gráfica do utilizador que é utilizada para criar, modificar e implantar canais. É ainda possível monitorizar toda a atividade proveniente dos canais, assim como navegar no armazenamento de mensagens.
- ***Mirth Connect Server Manager*:** pequena aplicação que fornece as definições para configuração do *Mirth Connect Server* e permite escolher a bases de dados pretendida para armazenar dados com informação relativa aos canais e mensagens.
- ***Mirth Connect Command Line Interface*:** ferramenta que proporciona uma maneira alternativa e leve de interagir com o servidor do *Mirth*. Não tem acesso a tantas funcionalidades do *Mirth Connect Administrator*, apenas às mais básicas.

De seguida, conforme experiências realizadas pela organização fundadora aquando o desenvolvimento deste *middleware*, são retratados os três principais componentes inerentes à sua arquitetura [70].

4.1.1 *Modelo de Interface*

Este modelo é essencial, pois foi pensado de maneira a definir todo o processo que as mensagens deverão de percorrer para serem recebidas, processadas e transmitidas, conforme os requisitos pretendidos pelo utilizador.

Como já foi visto na secção 2.3.1, a interface para as mensagens é designada de **canal** e é constituído por 4 componentes principais: **Conetor de origem**, **Filtro**, **Transformador** e **Conetor de Destino**.

Este modelo representa um dos elementos fundamentais para que a arquitetura desta plataforma fosse implementada de forma a proporcionar um bom funcionamento. Devido a esta arquitetura baseada em componentes, proporciona a separação entre os componentes que trabalham com o protocolo de transmissão (conectores de origem e destino) e os que lidam com o tratamento e processamento dos dados da mensagem (o filtro e o transformador).

Esta divisão das diferentes funções, possibilita que o filtro e a lógica de transformação da interface sejam desenvolvidos independentemente da lógica de transmissão. Através deste fator, é permitida a alteração dos tipos de conetores de origem e destino, sem envolver a modificação do filtro e a lógica do transformador.

Através desta importante abstração é garantida a flexibilidade em termos de conexão a sistemas externos, mantendo toda a lógica inerente ao processamento dos dados.

4.1.2 *Servidor*

O servidor desempenha duas funções importantes nesta arquitetura. Em primeiro lugar, é um container para os canais implementados, onde estabelece as conexões para os conetores e habilita os serviços de filtragem e transformação.

A arquitetura ESB implementa as noções inerentes de arquiteturas orientadas a serviços (SOAs) e é incorporada no servidor, pois fornece uma abstração dos recursos necessários para mensagens baseadas em serviços. Deste modo, os componentes arquitetónicos são retratados como serviços e são invocados por intermédio de um barramento de mensagens orientado a eventos. Isto permite flexibilidade no sentido em que os serviços não são integrados diretamente na arquitetura, podendo ser modificados durante o tempo de execução.

A capacidade de suportar chamadas de serviço síncronas e assíncronas é algo fundamental nesta arquitetura, porque permite o fluxo bidirecional de mensagens:

- Síncrono: uma resposta pode ser gerada no conetor de destino e retornada pelo conetor de origem para o sistema de envio original, caso a mensagem seja transmitida com sucesso.
- Assíncrono: não é necessária a confirmação do sucesso da mensagem, permitindo um processamento de mensagem mais rápido.

As mensagens recebidas de determinada fonte externa são colocadas no barramento, através de um conetor de origem designado de adaptador que representa um dos componentes do modelo ESB. Os conectores de destino também são representados como adaptadores individuais entre o barramento e um sistema externo. Assim que as mensagens chegam ao barramento, são codificadas em *XML*, num processo designado normalização, garantindo que todos os dados possam ser referenciados de maneira padrão. Através do encaminhador, capacita o sistema de decidir o destino de uma mensagem durante seu transporte. Todos estes elementos referenciados (adaptador e encaminhador), assim como os filtros e transformadores, são representados como serviços ligados a um barramento de mensagens. A figura 4.2 ilustra o que foi descrito, onde o sentido bidirecional reflete o sincronismo das chamadas aos serviços.

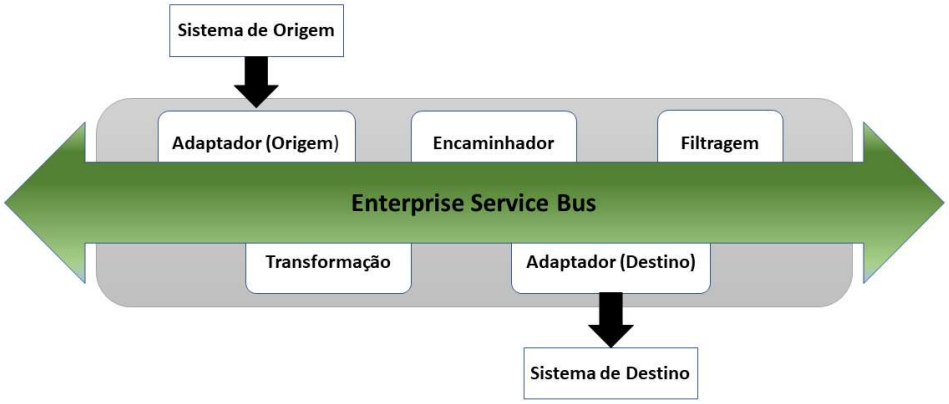


Figura 4.2: Exemplo de uma arquitetura ESB (adaptado de [6]).

Relativamente à segunda função do servidor, proporciona o armazenamento dos canais utilizados no ambiente de desenvolvimento. Isto é realizado usando o estilo *Model-View-Controller (MVC)* sob um sistema de bases de dados.

4.1.3 *Cliente*

Este componente retrata todo o ambiente de desenvolvimento de interface, que proporciona ao utilizador o desenvolvimento, teste, implementação e monitorização dos canais. É uma interface orientada a tarefas, ou seja, é permitida a configuração dos elementos individuais de um canal conforme uma lista de ações sensível ao contexto.

De maneira a suportar a adição de novos conetores, foi utilizada uma arquitetura plug-in, em que os pontos de extensão foram implementados tanto no servidor como no cliente. Assim, permite o carregamento dos módulos do conetor na inicialização. Através de uma API, os conetores permitem aceder às mensagens no barramento, enquanto encapsulam toda a lógica de transmissão inerente ao protocolo usado.

Um outro aspeto importante deste componente é a possibilidade do utilizador ter controlo sobre o filtro e lógica do transformador. De forma a disponibilizar maior flexibilidade ao utilizador ao criar regras de filtro ou etapas de transformação que não se encontram disponíveis na interface, pode usar *JavaScript* como linguagem de programação para a configuração pretendida. Esta linguagem contém uma extensão (designada *ECMAScript*) que fornece um meio de trabalhar com as mensagens *XML* normalizadas, através de uma sintaxe simples.

Relativamente aos transformadores, foram desenvolvidos para proporcionar a adição de novas etapas, de forma a manipular as mensagens recebidas. Estas etapas são constituídas por vários tipos, dos quais se destacam o mapeamento de elementos de dados das mensagens recebidas para variáveis globais, o tipo de etapa definida pelo próprio utilizador usando *JavaScript* e, por fim, o tipo de etapa do construtor, onde elementos de dados da mensagem de entrada são mapeados para elementos de uma estrutura de mensagem predefinida (por exemplo, conversão de uma mensagem HL7 versão 2 para versão 3).

4.2 PRINCIPAIS FUNCIONALIDADES

Como já foi evidenciado, o *Mirth* é composto por 4 componentes essenciais que constituem a sua arquitetura. No entanto, o *Mirth Connect Administrator* constitui interface gráfica principal com a qual o utilizador mais interage, criando, modificando e implementando canais.

Posto isto, de seguida são abordadas as funcionalidades essenciais relativas à configuração dos canais, organizadas pelos guias evidenciados na figura 4.3.

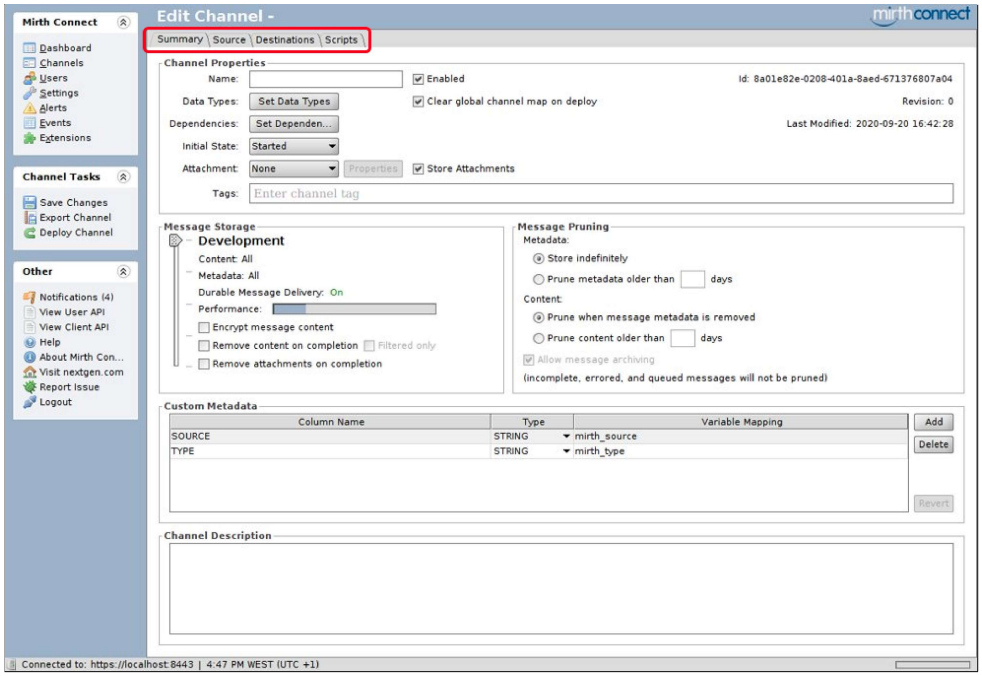


Figura 4.3: Interface para criação de canais do *Mirth Connect Administrator*.

4.2.1 *Summary*

Esta é a primeira secção para criação e desenvolvimento de um canal, onde se pode formular algumas configurações gerais. É constituída por três sub-secções principais: *Channel Properties*, *Message Storage* e *Message Pruning* (figura 4.3).

Relativamente à primeira, são de importante relevância as seguintes configurações:

- ***Name***: nome atribuído ao canal, devendo ser específico da sua função.
- ***Data Types***: opção que permite ao utilizador modificar as pré-definições do formato de dados que é recebido através da fonte. Por exemplo, por norma, aquando a criação, todos os canais assumem que os dados que chegam de uma fonte, assim como os que são enviados ao destino, estão no formato HL7v2.x. Assim serão feitas algumas verificações do formato de dados de cada mensagem, durante o seu percurso. Desta forma, neste passo existe a possibilidade de alterar o tipo de dados em cada etapa de verificação, ou até configurar para que nenhuma validação seja realizada. Este tipo de etapas designa-se de *inbound* e *outbound*.

É ainda proporcionado, conforme o tipo de dados selecionado, fazer o *parser* da mensagem de modo a obter os dados com mais precisão (tal como remover os *namespaces* de um ficheiro *XML*).

- ***Dependencies***: permite a configuração de dependências externas de bibliotecas, necessárias para a implementação do canal.
- ***Attachment***: possibilita a extração de partes de uma mensagem recebida, armazenando-as separadamente em memória, para posterior utilização no processamento.

A segunda sub-secção define a quantidade de conteúdo de uma mensagem que é retido na base de dados, ou seja, determina os dados que são armazenados, se encripta ou se elimina automaticamente o conteúdo, após a conclusão do processamento das mensagens.

O armazenamento destes dados determina a quantidade de informação que será apresentada ao utilizador, aquando o processamento das mensagens. Desta forma, existem 5 configurações possíveis:

1. ***Development***: Todos os dados são armazenados.
2. ***Production***: Preserva os dados essenciais como dados *raw*, metadados, conteúdo codificado, enviado e respondido, tal como o mapeamento efetuado. Apenas não armazena as respostas recebidas pelo conetor destino, relativamente à confirmação da receção de mensagens.
3. ***Raw***: Armazena apenas a mensagem de origem original, sem guardar meta-dados e variáveis mapeadas. Permite ainda reprocessar uma mensagem, quando necessário.
4. ***Metadata***: Apenas é armazenado o registo referente a cada mensagem, apenas para visualização do utilizador. Nesta configuração já não é permitido reprocessamento.
5. ***Disable***: Não são armazenados meta-dados, conteúdos ou anexos de mensagens. As filas de origem e de destino não são suportadas neste modo.

Quanto mais baixo o nível do modo a ser escolhido, menos informação é retida e, consequentemente, maior é a performance do canal, isto é, processa mensagens mais rápido e consume menos memória.

Por último, na sub-secção ***Message Pruning*** é onde o utilizador pode determinar após quanto tempo, as mensagens de determinado canal podem ser eliminadas, tanto os metadados como o seu conteúdo. Caso o armazenamento seja uma preocupação e exista uma ideia clara de quanto tempo atrás normalmente será necessário procurar mensagens processadas no canal, então torna-se útil a configuração desta funcionalidade de forma a limitar a quantidade de disco que as mensagens ocupam. Apenas permite a remoção de mensagens processadas ou filtradas.

4.2.2 *Source*

Neste guia, é onde são definidas as configurações necessárias para a receção de mensagens pelo canal. O painel principal da guia *source* define o tipo de conetor a ser estabelecido, no entanto, existem diversos tipos, cada um com as suas próprias configurações.

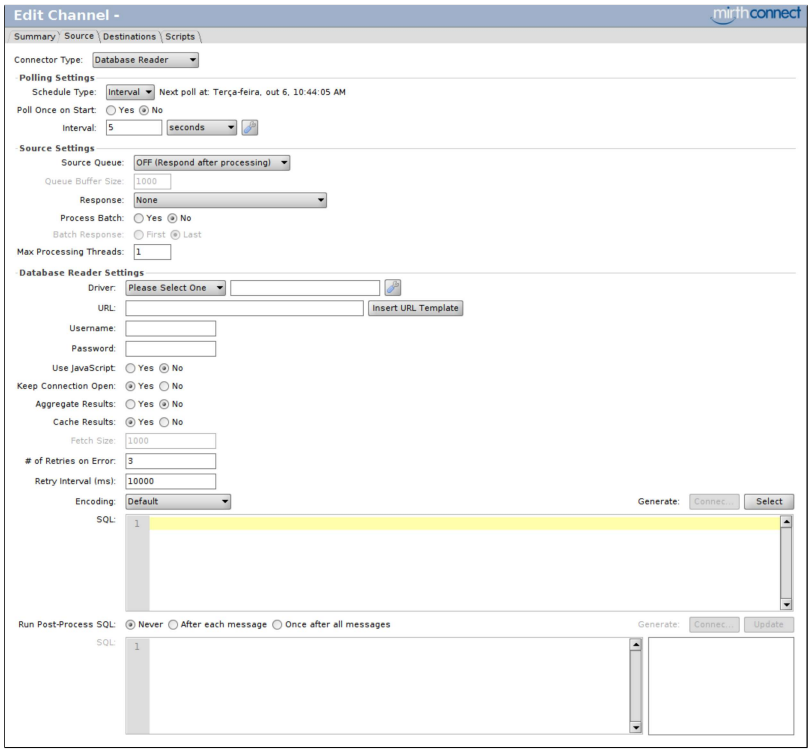


Figura 4.4: Interface do conetor de origem do *Mirth Connect Administrator*.

Desta forma, inicialmente vai ser abordada a secção *"Source Settings"* já que é sempre exibida, independentemente do tipo de conetor utilizado:

- ***Source Queue***: Define se a fila de origem está ativa, determinando o momento do envio da resposta da mensagem. Desta forma, caso a fila esteja desabilitada, a mensagem será processada e posteriormente será enviada a resposta, ou seja, a resposta é despoletada por cada mensagem individualmente. Caso a fila de origem esteja habilitada, as mensagens são enfileiradas e as respostas são enviadas imediatamente, antes do seu processamento.

- ***Queue Buffer Size***: Este campo é destinado ao anterior. Define o tamanho do *buffer* para a fila de origem, isto é, o número de mensagens que podem ser mantidas em memória, de uma só vez.

- ***Response***: Define a resposta enviada de volta ao sistema de origem, para cada mensagem. O utilizador pode optar por uma resposta definida no destino, por o valor de retorno do pós-processador, por uma resposta desenvolvida no *javascript writer* (designada de *map response*), por respostas automáticas ou por nenhuma. Relativamente às respostas automáticas, geram uma mensagem *ACK* para mensagens do tipo HL7 que servem para indicar a confirmação do recebimento ou para relatar numa falha (NACK). Existem 3 tipos de respostas automáticas:
  - *Before processing*: resposta gerada antes do processamento da mensagem (*SENT*).
  - *After source transformer*: resposta gerada depois do processamento da mensagem (estado do conetor de origem).
  - *Destinations Completed*: resposta gerada após o processamento da mensagem e depois da sua passagem pelo destino (*ERROR, QUEUED, SENT, FILTERED*).

- ***Process Batch***: Permite dividir um ficheiro em várias mensagens. Maioritariamente utilizado para ficheiros *CSV*, onde se pode optar por analisar cada linha ou componente como uma mensagem (opção *YES*), ou analisar o ficheiro completo (opção *NO*).

- ***Max Processing Threads***: É definido como 1 por padrão, o que significa que uma mensagem é processada de forma síncrona, de cada vez. Aumentar este valor, é usado *multithreading*, onde várias mensagens são processadas ao mesmo tempo, aumentando o consumo de memória.

Connector Type

O *Mirth* incorpora um vasto leque de diferentes tipos de conetores pré-estabelecidos, que permitem a ligação a diversos sistemas externos. Assim, existem dois tipos de conetores:

- ***Listeners*:** O tráfego é recebido por meio de protocolos, exigindo que a conexão seja estabelecida através de uma porta específica.

- ***Readers*:** conetores configurados para fazer a procura de dados ou permitir a execução de uma função específica, num intervalo de tempo definido. Ao contrário do anterior, a pesquisa de dados é realizada a cada período de tempo, definido pelo utilizador.

Na tabela 4.1, são especificados os principais conetores de origem e a correspondente descrição de cada um, sendo apresentada de forma breve as especificações mais relevantes.

Tipo de Conetor	Descrição
Channel Reader	Permite que um canal receba mensagens enviadas a partir de outro(s), dentro da mesma instância *Mirth*. É bastante utilizado para canais de teste ou para tratamento de erros.
DICOM Listener	É usado para receber mensagens por meio do protocolo *DICOM*, por exemplo, imagens transmitidas de dispositivos médicos de raios X. As mensagens aparecem como imagens codificadas em base64, com metadados em formato *XML* e um anexo com a imagem (o que obriga a ativação da opção *Attachment*, do guia inicial). A conexão é estabelecida por um endereço e uma porta local.
Database Reader	Coneta-se a uma base de dados relacional *SQL* externa através de um *driver*, permitindo realizar consultas *SQL*, lendo as linhas selecionadas que entram no canal no formato *XML*. O utilizador pode optar por executar a consulta, no pré-processador, em *SQL* ou através do desenvolvimento de código em *JavaScript*. Existe também o pós-processador, que permite executar operações sobre as mensagens, após terem sido processadas.
File Reader	Possibilita a leitura de ficheiros que se encontrem numa máquina local, especificando a diretoria, ou remotamente, através de protocolos adequados (*FTP*, *SMB*,etc). Depois de ler os ficheiros, existe a possibilidade de excluí-los, renomeá-los ou movê-los para uma outra diretoria.
HTTP Listener	Atua como um servidor *HTTP*, que espera solicitações de vários clientes remotos, através de uma porta local. As mensagens podem ser recebidas num formato em *XML* ou texto delimitado. O utilizador pode ainda personalizar a resposta enviada para o sistema emissor.
JavaScript Reader	São executadas funcionalidades desenvolvidas pelo utilizador. Normalmente é utilizado para conexões que não estejam disponíveis em nenhum dos outros leitores ou para monitorizar outros canais em execução. Isto é realizado através da execução de código escrito em *JavaScript*, podendo utilizar bibliotecas externas de *Java*.
TCP Listener	Escuta as mensagens que chegam através de uma conexão efetuada pela configuração de uma interface de rede e uma porta local. Esta conexão pode ser estabelecida para clientes ou para servidores externos. É também proporcionada a escolha de dois modos de transmissão: *Basic TCP* e *MLLP*.

Tabela 4.1: Especificação dos tipos de conetores da origem.

Source Filter

Este é um dos componentes que proporciona a filtragem de mensagens específicas para o processamento, conforme a lógica implementada pelo utilizador. Este filtro é bastante útil pois permite fazer a avaliação e seleção das mensagens a receber, com base no conteúdo da própria mensagem. Sendo que lógica inerente é idêntica para a origem e para o destino, será abordada com mais detalhe nesta sub-secção.

Assim que se cria uma nova regra, por *default* o tipo será *Rule Builder*. Esta configuração permite ao utilizador escolher a condição pretendida para valores definidos, e o *Mirth* gera o código de forma automática no *back-end*. Caso se pretenda um maior controlo sob o filtro, também existe a opção de alterar o tipo da regra para *JavaScript*, onde o utilizador desenvolve o seu código com base nas suas necessidades.

É possibilitada a introdução de várias regras e a modificação do operador com o qual a lógica será avaliada (*"AND"* ou *"OR"*, figura 4.5). A opção *field* é onde se insere o caminho do campo que se deseja avaliar, por exemplo, numa mensagem HL7 o caminho será algo como *"msg['MSH']['MSH.3']['MSH.3.1'].toString()"*.

Caso não exista um conhecimento o suficiente abrangente da estrutura deste tipo de mensagens, existe a possibilidade de colocar um exemplo de mensagem na coluna *Message Template*, e depois arrastar os campos pretendidos do estilo *drag-and-drop*, pela mensagem gerada na guia *Message Trees*.

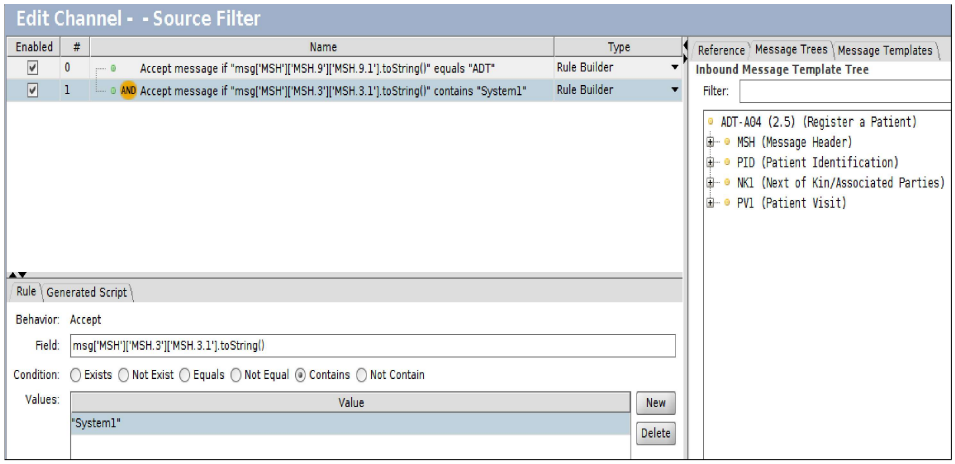


Figura 4.5: Exemplo de regras criadas no *Source Filter*.

No exemplo acima, foram criadas duas regras: a primeira, que apenas aceita a passagem de mensagens do tipo ADT e a segunda regra, que só permite mensagens cujo valor da aplicação fonte seja *"System1"*. Repare-se que o operador toma o valor *"AND"*, o que significa que ambas as regras necessitam de ser aprovadas para que uma mensagem não seja filtrada.

Caso as mensagens sejam filtradas, não são processadas através do transformador de origem ou qualquer que sejam os destinos, apenas são removidas de forma equivalente às mensagens processadas normalmente. Outro aspeto a ter em conta acerca deste componente é em relação às respostas. Se o utilizador configurar a resposta para algo diferente de *"Auto-Generate (Before processing)"*, a mensagem filtrada enviará um *NACK* como resposta, notificando que a mensagem foi rejeitada.

Source Transformer

Tal como o nome indica, este passo do ciclo de vida de uma mensagem, permite desenvolver código direcionado para a tradução de uma mensagem, ou seja, a forma como se deseja mapear os dados de maneira a transformar uma mensagem num outro formato. O *Mirth* executa este componente e o filtro como um único *script* para cada conetor.

Da mesma forma que o filtro permite a criação de várias regras, o transformador proporciona a criação de várias etapas, cada uma composta por diferentes tipos de transformação. Os principais tipos constituintes são:

- ***Mapper*** **:** Este tipo de transformação é utilizado para mapear um valor de determinado campo da mensagem, e atribuí-lo a variáveis *Map*:
  - *Channel Map* : É adquirido o valor e usado dentro da configuração do canal em questão. Em *javascript* definida por: *channelMap.put("variableName", value)*.
  - *Global Map* : Valor utilizado para qualquer outro canal, na mesma instância *Mirth*. Em *javascript* definida por: *globalMap.put("variableName", value)*.
  - *Response Map* : Valor usado para uma resposta personalizada, na interface atual. Em *javascript* definida por: *responseMap.put("variableName", value)*.

  As opções de configuração são bastante *user-friendly* (figura 4.6) e funcionam de forma idêntica ao *rule builder*, abordado anteriormente.

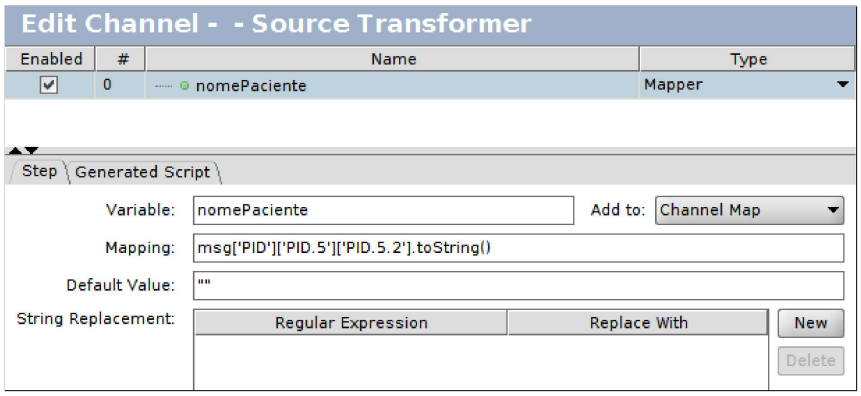


Figura 4.6: Exemplo de uma etapa do tipo *Mapper*.

Na figura acima, é ilustrado um exemplo de um mapeamento do campo *PID*-4.1 de uma mensagem HL7, para uma variável designada "nomePaciente". Sendo esta uma variável *channelMap*, pode ser chamada, dentro do canal, por "$(*'nomePaciente'*)".

- ***Message Builder*** **:** No caso do utilizador possuir uma estrutura bem definida da mensagem de saída, que possa ser facilmente mapeada, esta opção é a ideal. Possibilita o mapeamento direto dos dados da mensagem de entrada (*Inbound Message Template*), que chegam da fonte ao *Mirth*, para um modelo de mensagem de saída (*Outbound Message Template*).

  Como configurações disponíveis encontram-se o *Message Segment*, que define o caminho do campo da mensagem de saída para o qual se deseja atribuir o valor mapeado, o *Mapping*, valor a mapear da mensagem de entrada, e o *Default Value*, que define o valor padrão para o segmento da mensagem. Como anteriormente, estes valores podem ser preenchidos pelo estilo *drag-and-drop*.

  No exemplo abaixo apresentado (figura 4.7), está a ser mapeado um campo de uma mensagem *XML* ("nomePaciente") para um valor HL7 de saída (*PID*-5.2). A mensagem de saída é armazenada automaticamente numa variável reservada "*tmp*", que posteriormente pode ser usada no transformador de destino, isto se o modelo de mensagem de saída estiver definido.

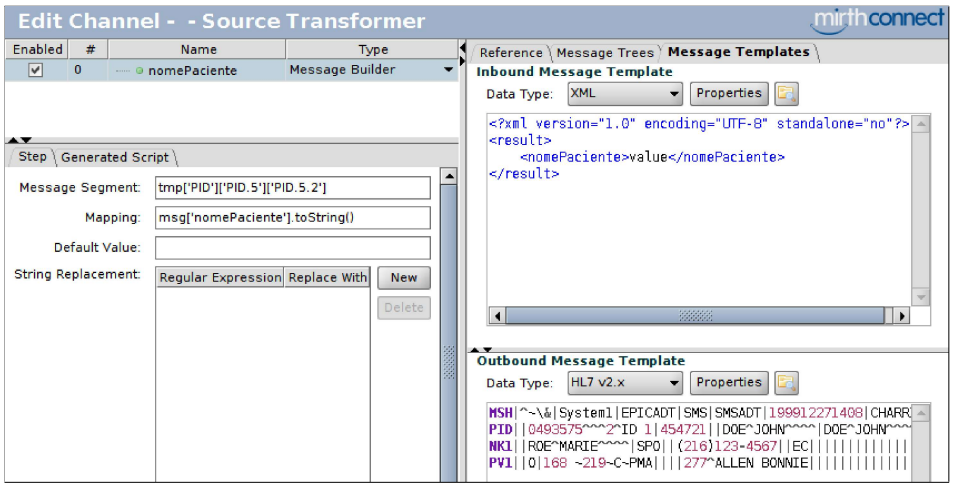


Figura 4.7: Exemplo de uma etapa do tipo *Message Builder*.

- ***JavaScript*** **:** Etapa idêntica à referida no filtro, que pode ser usada para programar código *JavaScript*, que realize o mapeamento quando as funcionalidades disponibilizadas na interface não forem suficientes ou se tornarem demasiado complexas.

- ***Destination Set Filter*** **:** Esta é uma etapa muito utilizada, que tem como principal finalidade de decidir com antecedência quais os destinos a excluir do processamento de mensagens, ou seja, a partir desta configuração é possível redirecionar mensagens para destinos específicos conforme condicionantes impostas pelo utilizador.

  A utilização deste tipo de controlo, que decide o destino de uma mensagem é um fluxo de trabalho válido, no entanto, caso existam diversos destinos, todos eles com filtros exclusivos, pode comprometer o desempenho do canal. Isto advém do facto dos dados da mensagem serem armazenados na base de dados, para cada conetor de destino.

Assim como nos canais e nas regras aplicadas nos filtros, as etapas estabelecidas nos transformadores podem ser exportadas ou importadas. Portanto, caso exista um transformador que possa ser útil em vários canais, não existe a necessidade de clonar o canal completo, basta apenas importar as etapas para os canais pretendidos.

4.2.3 *Destinations*

Nesta guia, é onde o utilizador pode configurar as funcionalidades de envio, com base em protocolos e informações estabelecidas, necessárias para o direcionamento das mensagens. Contrariamente ao que se sucede na origem, existe a possibilidade de estabelecer vários destinos simultaneamente ou utilizar diversos destinos para diferentes fluxos de trabalho. As opções de origem são praticamente idênticas, diferindo em alguns pormenores, como se poderá observar de seguida.

Tal como acontece na origem, o destino também contém configurações que se aplicam a todos os seus conetores, designadas *Destination Settings* (figura 4.8).

Figura 4.8: Configurações iniciais da secção *Destination*.

A primeira opção *Queue Messages*, definem o modo de entrada das mensagens para a fila conforme entram no destino, para serem processadas:

- ***Never*:** As mensagem fluem normalmente, sendo que no caso de não serem enviadas, serão marcadas como *ERROR* e o sistema passa para a mensagem seguinte.
- ***Failure*:** Coloca a mensagem na fila, no caso de falha, até que seja enviada com sucesso.
- ***Always*:** A mensagem vai para fila, antes de tentar ser enviada.

As duas últimas opções são benéficas caso o utilizador pretenda que a mensagem continue a ser enviada, no entanto, na hipótese do problema estar a associado ao conteúdo da mensagem, o estado será *QUEUED* indefinidamente.

Nas configurações da fila, permite especificar o número de tentativas de repetição automática e/ou um intervalo de tempo para executar essas tentativas. No caso das opções *Failure* e *Always* estarem ativas, o utilizador pode também ativar o *Rotate Queue*, que possibilita que as mensagens falhadas passem para o final da fila, de modo a evitar o atraso de mensagens futuras. Nesta opção, existe ainda outra configuração para habilitar o *multithreading*, caso se pretenda processar várias mensagens simultaneamente.

Em relação ao *Validate Response*, usa a lógica de validação de mensagens para determinar se o destino enviou uma mensagem válida ou não. Apenas se torna útil para confirmação de *ACK*'s de mensagens do formato HL7.

Por fim, a última opção, permite substituir o anexo inserido na origem, identificado através de um *token* associado à mensagem, e reinsere novamente com os dados atuais.

Connector Type

Os tipos de conetores disponíveis são idênticos aos anteriormente abordados, no entanto, as configurações de cada um diferem em alguns aspetos (tabela 4.2).

Tipo de Conetor	Descrição
Channel Writer	Permite a realização do envio de mensagens para outros canais internos, podendo ser útil para dividir o fluxo de trabalho de mensagens em vários canais.
Document Writer	Conexão que coloca os dados num modelo em *HTML* e converte para um documento *PDF* ou *RTF*. Estes documentos podem ser gravados numa pasta ou armazenados como um anexo de mensagem.
Database Writer	Possibilita a conexão a uma base de dados *SQL* externa e a execução de uma *query*, baseada em *SQL* ou *JavaScript*. Para aceder, basta selecionar o *driver* da base de dados e introduzir os dados de autenticação.
File Writer	Conetor que proporciona o armazenamento de ficheiros através da utilização de vários protocolos (os mesmos usados pelo *file reader*). Permite a atribuição de um nome para o ficheiro criado e de um tempo máximo limite que uma conexão pode ficar inativa ou não usada, antes de ser fechada.
HTTP Sender	Conexão efetuada através do URL pretendido. Pode ser escolhido o método (*POST*, *GET*, *PUT* e *DELETE*) e definir os *headers* (pares nome-valor, como "Autorização":<*token*>), necessários à solicitação. Por fim, encontra-se uma caixa de texto, onde o utilizador insere o conteúdo da mensagem a enviar.
JavaScript Writer	Idêntico com as opções de JavaScript do filtro e transformador de origem, permite a escrita manual do código, construindo a funcionalidade de envio pretendida pelo utilizador.
TCP Sender	Propicia a abertura de uma nova conexão de um cliente *TCP* e envia mensagens através da mesma. O utilizador pode optar por manter uma conexão aberta e, em caso afirmativo, por quanto tempo. Os modos de transmissão configuráveis , tal como para os conetores de origem, podem ser do tipo *MLLP* ou *Basic TCP*.

Tabela 4.2: Especificação dos tipos de conetores da destino.

Destination Filter

Funcionamento idêntico ao *source filter* em relação à lógica implícita, no entanto, diferentes filtros podem ser aplicados em destinos distintos. No caso do utilizador pretender enviar determinadas mensagens para um destino, e outras para outro, pode ser mapeado um valor no conteúdo da mensagem representativo de cada destino. Caso a configuração da resposta esteja definida para quando o transformador de origem terminar, é enviado de volta um *ACK*, mesmo que a mensagem seja filtrada posteriormente.

Este filtro pode ser substituído pela etapa do transformador de origem, designada *"Destination Set Filter"*.

Destination Transformer

Tal como anteriormente, operam de forma muito semelhante ao *Source Transform*, com a exceção que pode existir um transformador atribuído a cada destino. Depois das mensagens serem filtradas, se necessário, este componente permite enviar dados processados de diferentes formas para diversos destinos.

Destination Response

Contrariamente aos componentes da origem, para o destino existe a noção de respostas. Esta configuração aplica o mesmo conceito inerente aos transformadores, mas o seu processamento apenas é despoletado após a conclusão do destino e caso uma resposta seja retornada ao destino.

Desta forma, para os destinos que retornam uma resposta, através de um método interno do *Mirth* designado *"response.getMessage()"*, é possível fazer a análise do conteúdo da resposta. Para o processamento da resposta, tal como no transformador, podem se criar etapas com a lógica necessária para o que o utilizador pretender. O formato da mensagem de resposta esperado, pode ser definido no guia *Summary*, na opção *Set Data Types*. Por exemplo, se as respostas de destino forem do tipo HL7 *ACK*, é necessário definir a configuração de entrada como HL7 v2.x.

4.2.4 *Scripts*

Nesta guia, é possibilitado ao utilizador a execução e implementação de *scripts*, em determinados pontos do fluxo de trabalho de um canal. Todo o código que o utilizador pode implementar para ajustar às suas necessidades, é desenvolvido em *JavaScript* versão 6 (*ECMAScript 6*), usando o mecanismo *Rhino*, que permite dar acesso a classes e bibliotecas *Java* [71]. Por norma, independentemente do *script*, no final todos tem uma instrução *return*, que tanto pode estar associada a uma mensagem como a uma lista de mensagens.

Desta maneira, existem quatro fases do canal onde existe a possibilidade de executar um *script*:

- ***Deploy Script*:** É executado apenas uma vez, antes da inicialização do canal.
- ***Preprocessor Script*:** É executado uma vez para cada mensagem, após o conetor de origem receber uma mensagem e depois do *attachment handler* opcionalmente extrair os dados, mas antes da mensagem ser filtrada ou processada. A sua principal funcionalidade é a modificação de uma mensagem de entrada.
- ***Postprocessor Script*:** É executado para cada mensagem, após a conclusão do envio das mensagens para todos os destinos, mas antes da resposta do conetor de origem de volta ao sistema que originou a mensagem. Este *script* tem acesso às respostas de todos os destinos executados e pode retornar uma resposta personalizada, utilizada pelo conector de origem.
- ***Undeploy Script*:** É o oposto do *Deploy*, em que é executado após o término de funcionamento do canal.

5

TECNOLOGIAS DE DESENVOLVIMENTO

Durante o estudo e desenvolvimento de uma arquitetura que proporcionasse as condições determinantes para monitorização e avaliação das métricas foram necessárias a utilização de algumas ferramentas essenciais.

Neste enquadramento, este capítulo destina-se à apresentação das tecnologias fundamentais que auxiliaram o desenvolvimento e a implementação de uma infraestrutura que inviabilizou a análise dos indicadores em ferramentas de integração.

5.1 BASES DE DADOS

Um dos métodos mais utilizados para envio e receção de dados, de forma a testar o funcionamento da plataforma, foi a utilização de bases de dados. Por forma a abranger as principais tecnologias deste tipo, foram usadas bases de dados relacionais e não relacionais:

- **Bases de Dados Relacionais:** Caracterizadas por conjuntos de elementos de dados, estritamente organizados em tabelas formalmente descritas, onde os dados podem ser acedidos de diversas maneiras. As tabelas são constituídas por colunas, relacionadas com a categoria dos dados (atributos), e linhas, que caracterizam as instâncias dos dados para as categorias correspondentes.

  A ligação existente entre várias tabelas, define o nível de relacionamento implícito em cada uma e determinadas restrições inerentes aos atributos. A maioria das bases de dados relacionais usam a linguagem de programação *SQL* (excerto de código 5.1) para aceder e manipular os dados que estão armazenados [72].

```
SELECT Salario
FROM Empregados
WHERE Idade BETWEEN 25 AND 30
GROUP BY Categoria
```

Excerto de código 5.1: Exemplo de uma *query SQL*.

5.1.1 *MySQL*

MySQL consiste num *Sistema de Gestão de Bases de Dados Relacional* (*SGBDR*) *open-source*, desenvolvido e lançado pela primeira vez em 1995 pela empresa *MySQL AB*, sendo atualmente propriedade da organização *Oracle*. É uma tecnologia escrita em C e C++, compatível com todos os sistemas operativos principais.

Emprega o conceito de armazenamento de dados em tabelas e linhas, usando *SQL* para aceder e manipular dados. As operações mais utilizadas através desta linguagem são: *SELECT*, *UPDATE*, *INSERT* e *DELETE*.

Foi utilizada esta tecnologia, pois é das bases de dados relacionais mais usadas e devido à sua característica importante de ser bastante escalável. A ferramenta escolhida para lidar com a administração e gestão das base de dados utilizadas, designa-se de *MySQL Workbench*, devido à sua utilização prática e gratuita.

5.1.2 *Oracle Database*

Oracle Database (*Oracle DB*) foi lançado pela primeira vez em 1977, sendo apontado como uns dos sistemas de gestão de bases de dados relacional mais utilizados [64]. Tal como a maioria deste tipo de sistemas de gestão de bases de dados, baseia-se na linguagem de programação *SQL*, utilizando comandos como os anteriormente descritos.

No entanto, a *Oracle DB* recorre a uma extensão *PL*/*SQL*, linguagem procedural da *Oracle* que se estende à linguagem *SQL*, destinada à manipulação de dados. Resumidamente, os blocos de código são processados por mecanismos *PL*/*SQL*, que filtram os comandos *SQL* e são enviados individualmente para um módulo *SQL Statement Executor*, onde são aplicadas as operações sob os dados [73].

Esta tecnologia foi utilizada devido à conformidade com as bases de dados que são usadas nos possíveis centros hospitalares onde eventualmente poderão ser realizados os testes em ambiente real. É de salientar que o *IDE* escolhido para administração e gestão, denomina-se de *Oracle SQL Developer*, por ser uma ferramenta da *Oracle* bastante *user-friendly*.

- **Bases de Dados Não Relacionais:** Reconhecida por não ser uma ferramenta, mas sim uma metodologia composta por várias ferramentas interdependentes e concorrentes. Ao contrário das bases de dados relacionais, é capaz de armazenar dados não estruturados como imagens ou documentos, e o seu armazenamento é baseado em esquemas que não são fixos, isto é, não guarda os dados em tabelas.

Esta característica fornece uma alta flexibilidade ao nível da inserção ou exclusão de determinado atributo da base de dados. De uma maneira geral, o método de armazenamento é realizado conforme chaves de identificação e os dados encontram-se nas chaves atribuídas.

O armazenamento dos dados numa base de dados não relacional pode ser efetuado através de 4 principais tipos, *Key-Value*, *Document*, *Column/Field* e *Graph-Oriented*. Todas as diferentes bases de dados relativas aos tipos anteriormente mencionados, possuem as suas próprias funções e modelos de dados [74].

5.1.3 *MongoDB*

MongoDB é uma base de dados do tipo *Not Only SQL* (*noSQL*), orientada a documentos. Apresenta um esquema de alta flexibilidade e todos os seus dados são armazenados no formato de *JavaScript Object Notation* (*JSON*). Como já mencionado, esta base de dados guarda os documentos em coleções ao invés de tabelas. Tal arquitetura oferece suporte para *sharding*, que se relaciona com um processo de armazenamento de dados em máquinas diferentes e a capacidade de processá-los, conforme o volume dos dados aumenta [75].

Neste contexto, cada documento (excerto de código 5.2) presente numa coleção é provido de um único número de identificação, atribuído automaticamente, tornando a pesquisa mais rápida e eficaz. São caracterizados por possuírem um esquema dinâmico, ou seja, os documentos de uma determinada coleção não necessitam de ter exatamente o mesmo conjunto de campos.

```
{
    "_id" : ObjectId("5f9b0aecfaed1c5"),
    "nome" : "Catarina",
    "idade" : 29
}
```

Excerto de código 5.2: Exemplo de um documento guardado no *MongoDB*.

A principal metodologia para as operações de leitura e escrita são os métodos CRUD: *Create*, *Read*, *Update* e *Delete*. No excerto de código 5.3 exemplificado, é realizada uma operação de leitura aos documentos de uma coleção "*users*", que retorna o nome e a idade de pessoas que vivam em Braga.

```
db.users.find(
    { "Cidade": "Braga" },
    { "Nome": 1, "Idade": 1}
)
```

Excerto de código 5.3: Exemplo de uma consulta no *MongoDB*.

A utilização desta base de dados advém do facto de se destacar pela sua facilidade de uso e alto desempenho quanto à realização de consultas, algo bastante relevante para o desenvolvimento deste trabalho.

5.2 DOCKER

Através da utilização de tecnologias de virtualização são proporcionados vários benefícios numa infraestrutura, como a redução dos custos de *hardware*, facilidade quanto à migração de ambientes, rápida distribuição de aplicações por várias máquinas e melhor administração e manutenção dos serviços.

O *Docker* é uma tecnologia de virtualização, que permite a implementação de aplicações através de um ambiente execução baseado em *containers*. Um *container* permite empacotar uma aplicação com todas as suas dependências e executá-la numa outra máquina, independentemente das configurações que a máquina possa ter. Ao contrário da máquina virtual, em vez de criar um sistema operativo virtualizado inteiro, esta ferramenta permite que as aplicações usem o mesmo *kernel* do sistema operativo onde estão a ser executadas (figura 5.1), permitindo um aumento do desempenho e reduzindo o tamanho da aplicação [7].

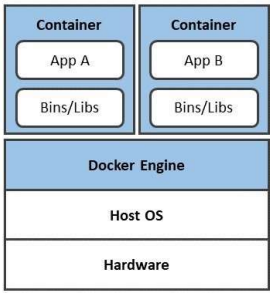


Figura 5.1: Arquitetura relativa à tecnologia *Docker* (adaptado de [7]).

Quando o *Docker* é iniciado, é criada uma interface de rede virtual que gera um intervalo de endereços *IP* virtuais e novos *containers* são criados dentro desta rede. Esta "ponte"permite que os containers comuniquem entre si.

Neste âmbito, existem 3 conceitos essenciais a ter em conta relativamente ao *Docker* [76]:

- ***Docker Image***: modelos binários de apenas de leitura, que permitem formar uma base para construir a raiz de um sistema de ficheiros de um *container*, de determinada aplicação.
- ***Dockerfile***: documento de texto que contém todos os comandos ou instruções necessárias para a execução de um *software*, tendo em conta as suas dependências. Uma imagem é montada através dos comandos existentes neste componente.
- ***Docker Compose***: ferramenta que permite a execução e implementação de aplicações em vários *containers*. É usado um ficheiro *YAML*, onde é realizada a configuração dos serviços necessários. É um mecanismo prático porque através de um único comando, são criados e inicializados todos os serviços de uma dada configuração.

Esta ferramenta, através do comando "*docker stats*", permite a visualização de estatísticas, em tempo real, do uso de recursos dos containers ativos [76]. Estes recursos são relativos ao uso de CPU e memória, bem como o total de *bytes* de operações *input/output* realizadas pela rede e no disco (ou sistema de ficheiros do *container*).

Esta tecnologia é essencial ao desenvolvimento do trabalho desta dissertação, pois proporciona a execução de inúmeras aplicações na mesma máquina, com alta performance, possibilitando montar ambientes de teste que podem ser distribuídos e utilizados em diferentes máquinas.

5.3 JMETER

JMeter é um *software open-source*, criado pela organização *Apache Software Foundation*, com principal funcionalidade de realizar testes e medir o desempenho de sistemas de *software*. É uma ferramenta desenvolvida em Java que inicialmente foi projetada apenas para fazer testes a aplicações *web*, mas foi expandido para outras funções de teste [77].

Também designado de ***Apache JMeter***, pode ser usado para simular diferentes tipos de cargas persistentes ao longo do tempo num servidor, grupo de servidores e até em redes por forma a analisar as condições máximas que um sistema suporta.

Possui uma interface gráfica abrangente, auxiliando na criação de planos de teste e configuração de elementos. É uma ferramenta gratuita independente do sistema operativo usado e pode ser usado para executar testes funcionais a aplicações [77].

Para criar um teste de carga no *JMeter* é preciso um plano de teste, que é constituído por vários componentes. Estes componentes são chamados de elementos, onde cada um é concebido para uma finalidade específica [78]:

- ***Thread Group*:** tal como indica o elemento, especifica um aglomerado de *threads* e um período de aceleração. Cada *thread* simula um utilizador usando a aplicação em teste, e o período de aceleração carateriza o tempo de criar todas as *threads*.

- ***Samplers*:** Normalmente os utilizadores virtuais são criados com a finalidade de geração de carga no sistema, como se fossem utilizadores reais. Este elemento estabelece o tipo de solicitação que é efetuada ao servidor por parte desses utilizadores. Esta função é considerada como o tipo de ação que o utilizador vai realizar. Estes pedidos podem ser *HTTP*, *FTP*, *JDBC* e muitos outros como poderemos verificar pelos protocolos apresentados a seguir.

- ***Listeners*:** exibe os resultados da execução dos testes efetuados. Podem mostrar os resultados em vários formatos: gráficos, tabelas, ficheiro de *logs* e até pode apresentar uma árvore de resultados (vários testes) em *HTML*.

- ***Configuration Elements*:** são usados para configurar ou modificar as solicitações dos *samplers* feitas no servidor. Esses elementos podem ser adicionados para todos os níveis de *samplers* que se pretenda configurar.

Esta ferramenta armazena todos os planos de teste num formato *XML*, o que para gerar um plano, o utilizador consegue realizá-lo através de um editor de texto. Não obstante, esta plataforma possibilita a realização de testes de carga de trabalho e desempenho para inúmeros tipos de tecnologias/protocolos.

Este sistema foi utilizado devido à sua caraterística de geração de carga suficiente para sobrecarregar o *Mirth*. Também viabiliza conexões via *TCP* e disponibiliza a configuração de seleção de vários clientes virtuais, ou *threads*, para enviar mensagens em simultâneo, o que para o desenvolvimento deste trabalho é algo fundamental.

6

ARQUITETURA E IMPLEMENTAÇÃO

Tal como já foi referenciado anteriormente, este trabalho tem como principal foco o estudo das principais métricas de desempenho da plataforma *Mirth Connect* de modo a obter um fluxo ideal relativamente ao processamento de mensagens.

Com o principal intuito de responder ao objetivo principal da dissertação, neste capítulo serão abordadas as tecnologias utilizadas, assim como a arquitetura ideal. Estas metodologias e arquitetura serão as ideais para a análise e a monitorização da plataforma de integração, de forma a tornar percetível os principais indicadores que causam desestabilização do seu desempenho. Posteriormente, são apresentados todos os casos de teste efetuados ao *middleware*, num ambiente de teste, que permitiram demonstrar as condições com as quais o fluxo de mensagens é afetado.

6.1 ARQUITETURA UTILIZADA

Como foi referenciado no capítulo 5.2, usando a tecnologia de virtualização *Docker* é permitida a criação de um ambiente de teste, possibilitando a execução de vários sistemas de *software* na mesma máquina. Esta caraterística permite uma enorme flexibilidade relativamente a qualquer arquitetura implementada, no sentido em que posteriormente a um ambiente de teste facilmente pode ser testada e migrada para um ambiente real.

Neste contexto, antes de se proceder à realização de testes, inicialmente foi necessário montar uma infraestrutura no *Docker*, capaz de acolher todas as tecnologias necessárias que proporcionam o bom funcionamento do *Mirth Connect*. Para isso, o melhor método de o fazer foi utilizar o ficheiro *Docker Compose*, começando pela criação dos *containers* precisos e de seguida a verificação da comunicação entre eles.

Desta forma, como apresentado na figura 6.1, para o funcionamento do motor de integração são necessários 4 *containers* fundamentais:

- Base de dados de *logs*;
- *Mirth Connect*;

- Bases de dados fonte que envia mensagens;
- Bases de dados destino que recebe mensagens;

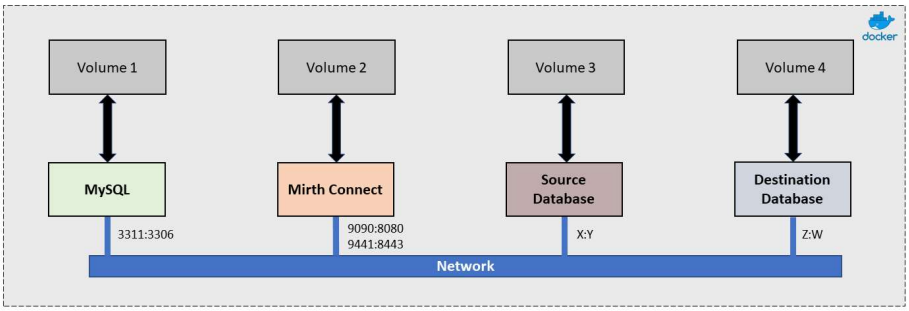


Figura 6.1: Arquitetura dos *containers* utilizados através do *Docker*.

Para o armazenamento do histórico de mensagens e outros eventos associados de fluxo de atividade, o *Mirth* dispõe de várias bases de dados relacionais (*MySQL*, *PostgreSQL*, *SQL Server*, *Oracle*, *Apache Derby*) por forma a guardar toda esta informação. Assim, foi utilizada uma base de dados *MySQL* durante todos os testes realizados.

Como se pode observar na arquitetura descriminada na figura 6.1, existem mais dois *containers* de bases de dados que se destinam a receber e enviar mensagens. O seu tipo não foi definido, porque ao longo de todo o trabalho desenvolvido foram usadas diferentes bases de dados, que serão apresentados na secção seguinte.

Em relação ao *container* do *Mirth Connect*, sendo o núcleo desta arquitetura, vai ser descrito com mais detalhe o seu processo de virtualização através do *Docker*. Para tal, é explicado o excerto relativo a este *container*, no ficheiro *docker compose* (figura 6.1).

Como se pode observar pelo excerto do ficheiro *YAML* apresentado, por forma a executar todos os ficheiros compreendidos pela instalação e execução do *Mirth*, foi necessário usar as chaves "*build*"e "*image*". Para a primeira chave é fornecido o nome da pasta que contém o *dockerfile*. Este é um ficheiro onde são definidas as opções de configuração e comandos necessários para montar a imagem correspondente (*fabrom/mirth-connect*). Isto engloba eventos como a atribuição de variáveis de ambiente e criação de diretorias precisas para o sistema de *software* no *container*, bem como comandos para a inicialização do servidor do *Mirth*, já abordado na sub-secção 4.1. A versão utilizada do *Mirth* durante todo o trabalho desenvolvido foi a 3.8.0.

De seguida, com o uso da próxima chave "*volumes*", é possível copiar ficheiros para determinadas diretorias do host do *Docker*. O primeiro ficheiro *mirth.properties*, contem principalmente o estabelecimento das configurações da base de dados de *logs* que o *Mirth*

```
nextgen:
  build: mirth-connect
  image: fabrom/mirth-connect
  container_name: nextgen
  volumes:
    - "./mirth-connect/mirth.properties:/opt/mirth-connect/conf/
      mirth.properties:rw"
    - "./data/envia/:/envia/:rw"
    - "./data/recebe/:/recebe/:rw"
  ports:
    - "9090:8080"
    - "9441:8443"
  links:
    - nextgendb
  depends_on:
    - nextgendb
```

Excerto de código 6.1: *Container Mirth* definido no ficheiro *Docker Compose*.

utiliza. Relativamente às outras duas pastas (*envia* e *recebe*), permite a colocação/obtenção de mensagens de/para as diretorias do *host* do *Docker*. Isto é necessário devido ao *Mirth* estar virtualizado e só se poder aceder a pastas desse *host*.

Posteriormente, relativamente à chave "*ports*", mapeia as portas pelo qual o *Mirth* irá operar. Tendo o motor de integração uma arquitetura servidor-cliente, é necessário o uso de uma rede e a definição de portas de forma a estabelecer a comunicação. Para tal, aquando a execução do ficheiro *Docker Compose*, por *default*, é configurada uma interface de rede. Através do endereço físico IPv4 da interface criada, é então possível definir portas que são necessárias para a execução do *Mirth*.

Por fim, a chave "*link*", tem como principal papel associar o *container Mirth* a outro serviço, que neste caso trata-se da Base de dados de *logs MySQL*. A chave "*depends_on*", permite que quando o ficheiro *Docker Compose* for executado, inicie primeiro o serviço *MySQL* e de seguida o *Mirth*.

6.2 ARQUITETURA DE TESTE

Ao longo de todo o processo de desenvolvimento deste trabalho, esta arquitetura foi sofrendo bastante alterações, conforme as necessidades para a análise da performance do fluxo de mensagens. A montagem desta arquitetura foi um dos passos mais importantes desta dissertação, pois através da sua implementação são fornecidos resultados cruciais que permite averiguar e avaliar os pontos de falha.

A arquitetura exibida na figura 6.2 representa a estrutura lógica utilizada, onde é demonstrado todo o caminho percorrido por uma mensagem desde o sistema fonte, passando pelo *Mirth*, até ao sistema destino.

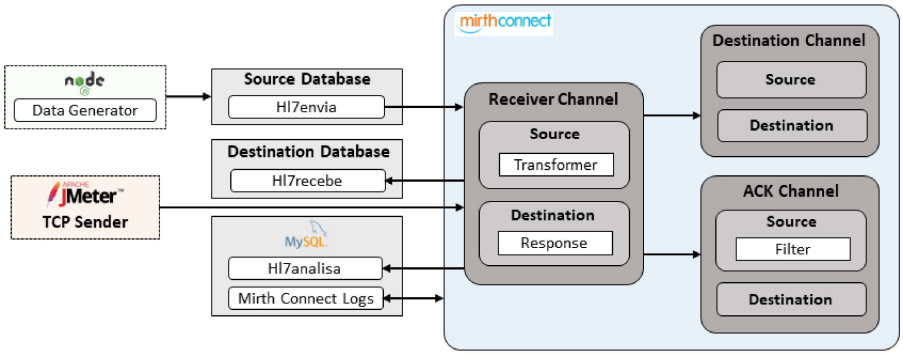


Figura 6.2: Arquitetura de canais utilizada no motor de integração *Mirth Connect*.

De seguida, são caracterizados todos os canais demonstrados, retratando os restantes componentes associados.

6.2.1 *Receiver Channel*

Este canal é responsável por receber dados ou mensagens de uma fonte externa, transformar os dados para um formato pretendido pelo utilizador e posteriormente enviar para um destino. Este é considerado o canal principal, pois é onde se encontra implícita toda a lógica que auxilia a análise do caminho, bem como os tempos das mensagens.

Neste contexto, inicialmente para a criação de diferentes mensagens, foi desenvolvido um *script* na linguagem de programação *Node.js*, que faz a geração de dados aleatórios de uma mensagem. Como se pode observar pelo *script* exposto no Apêndice A (excerto A.1), é feita a inserção de 400 linhas numa base de dados MySQL de 30 em 30 segundos. Este *script* é também executado para outras bases de dados, como Oracle, SQL Server, Postgres, SQLite e MongoDB, de maneira a serem abordados diferentes tipos.

hl7envia	
Atributo	**Tipo**
ID	int
Origem	varchar(45)
Destino	varchar(45)
MsgId	int
NomeFich	varchar(45)
Evn	varchar(45)
DataGer	timestamp
Episodio	int
Status	int

Tabela 6.1: Caracterização da tabela hl7envia.

A tabela 6.1 é constituída por atributos essenciais à constituição de uma mensagem HL7 v2 como os campos *Origem* (identificação do sistema que enviou os dados), *Destino* (identificação do sistema que irá receber os dados), *Evn* (para comunicar informações necessárias do evento do *trigger* a aplicações recetoras, figura 2.2) e *Episodio* (identificação de uma possível visita a um paciente). Relativamente ao atributo *MsgId*, funciona como um identificador que facilita a ordem das mensagens; *NomeFich*, para nomear a mensagem; *DataGer*, gera a data em que os dados foram criados para posterior comparação com outras datas, e o campo *Status*, onde o seu valor pode ser 1 ou 0 significando se a mensagem foi ou não processada, respetivamente.

A tabela relativa ao *Mirth Connect Logs*, é criada aquando o *deploy* da arquitetura montada no *Docker*. É o local onde são armazenadas todas as informações e meta-dados acerca do fluxo de mensagens, assim como dados relativos às configurações dos canais criados.

Após a inserção dos dados na base de dados pela execução do *script*, os valores dos campos de dados de cada linha são lidos através do *Mirth*, mais especificamente pelo canal *Receiver*. Para tal, foi escolhido o tipo de conetor *Database Reader* e, de seguida, o URL do *driver JDBC* da base de dados utilizada. A *query SQL* a seguir apresentada, é executada conforme um *pooling* de 2 segundos, para ler as mensagens.

```
SELECT hl7envia.ID AS teste_id,
       hl7envia.Origem AS hl7envia_origem,
       hl7envia.Destino AS hl7envia_destino,
       hl7envia.MsgId AS hl7envia_msgid,
       hl7envia.Nomefich AS hl7envia_nomefich,
       hl7envia.Evn AS hl7envia_evn,
       hl7envia.DataGer AS hl7envia_datager,
       hl7envia.Episodio AS hl7envia_episodio,
       hl7envia.Status AS hl7envia_status
FROM hl7envia
WHERE hl7envia.Status=0;
```

Excerto de código 6.2: Query do conetor de origem relativo ao *Receiver Channel*.

Pela *query* do excerto de código 6.2, é feita a consulta de todos os atributos da tabela *hl7envia* e cada um é mapeado para uma variável interna do *Mirth*. Estas variáveis apenas podem ser acedidas através de código desenvolvido pelo utilizador, como se poderá observar mais à frente. Note-se que a consulta é realizada para todas as linhas que contém o campo *Status* a 0, pois constituem as linhas que ainda não foram processadas.

```
UPDATE hl7envia
SET Status = 1 WHERE ID = ${teste_id}
```

Excerto de código 6.3: *Query* do *Post-Processor* do *Receiver Channel*.

Posteriormente, no pós-processador foi definida a *query* acima (excerto 6.3). Este passo foi efetuado no sentido de atualizar o atributo *Status* para o valor 1, após uma mensagem ser enviada para o sistema destino. Desta maneira, todas as mensagens que são processadas não são repetidas.

Por *default*, o *Mirth* no conetor *Database Reader* apenas contém bases de dados *SQL*. Assim, caso se pretenda incluir uma base de dados do tipo *noSQL*, é necessário adicionar o *driver* respetivo e depois, através do desenvolvimento de código em *JavaScript*, aceder à base de dados. Desta forma, para ser possível a comparação dos dois tipos, foi utilizada uma base de dados *Mongo*. Tal como anteriormente, foi criada a conexão e uma consulta à base de dados (excerto de código 6.4), que é executada conforme um *pooling* estabelecido.

```
try {
  //Estabelece a conexao
  var mongoClient = new Packages.com.mongodb.MongoClient("ipadress", port);
  var db = mongoClient.getDatabase("dbteste");
  var collection = db.getCollection("hl7envia");
  var list = new java.util.ArrayList();

  //Criacao do iterador e cursor com regra de consulta
  var iterador = new com.mongodb.client.FindIterable(new org.bson.Document());
  var cursor = new com.mongodb.client.MongoCursor(new org.bson.Document());
  var rule_filter = Packages.org.bson.Document.parse(
                                        JSON.stringify({"STATUS":"0"}));
  iterador = collection.find(rule_filter);
  cursor = iterador.iterator();

  //Iteracao das mensagens, mudando o valor do status e adicionando a lista
  while (cursor.hasNext()){
      var document = new org.bson.Document();
      document = cursor.next();
      var json_doc = document.toJson();
      var doc = JSON.parse(json_doc);
      var id_msg = Packages.org.bson.Document.parse(
                                    JSON.stringify({"_id": oJ._id}));
      var status = Packages.org.bson.Document.parse(
                                    JSON.stringify({ $set : {"STATUS": "1"}}));
      collection.updateOne(id_msg, status);
      list.add(cursor.next());
    }
  return list;
} finally {
    if (mongoClient) mongoClient.close();
}
```

Excerto de código 6.4: Excerto de código para conexão ao MongoDB.

No excerto acima apresentado, é criada a conexão é base de dados, e de seguida criados os iteradores e cursor necessários, conforme a *API* do *MongoDB*, para efetuar a consulta aos documentos da coleção. Note-se que, tal como anteriormente, a consulta é realizada para mensagens com o campo *status* igual a 0 e, posteriormente, o seu valor é atualizado para 1.

Após a consulta, cada documento é adicionado a uma lista que é devolvida ao *Mirth* para futuro processamento.

Um outro tipo de conetor utilizado foi o *TCP*, com modo de transmissão *MLLP*. Para o envio de mensagens foi utilizada a ferramenta *JMeter* (sub-capítulo 5.3). Inicialmente, para a configuração deste *software*, foi criado um *Thread Group* com um *TCP Sampler*. Como o *JMeter* não possui o protocolo de transporte *MLLP*, foi necessário utilizar um *plugin*[1]. De seguida, foi introduzido o endereço *IP* e a porta de conexão ao *Receiver Channel*, e por fim foi inserido um *template* de uma mensagem HL7 v2, com o campo corresponde do *msgid* a ser iterado, conforme cada mensagem enviada.

Seguidamente aos dados serem consultados, tanto para bases de dados *SQL* como para *noSQL* ou para mensagens recebidas via *TCP*, toda a informação obtida passa a ser tratada no transformador deste canal.

Relativamente aos dados originários de bases de dados, como cada linha consultada atinge o canal no formato *XML*, foi necessário fazer a sua transformação para uma mensagem HL7 v2. Como apresentado no sub-capítulo 4.2.2, através da introdução do *template* em *XML* com os atributos definidos da tabela *hl7envia* e o *template* de uma mensagem HL7v2, foi possível fazer a transformação dos dados recebidos numa mensagens HL7 (figura 6.3).

Em relação às mensagens provenientes do conetor *TCP*, o processo realizado foi idêntico, mas ao invés do formato *XML*, o mapeamento é efetuado aos elementos de um *template* HL7 v2.

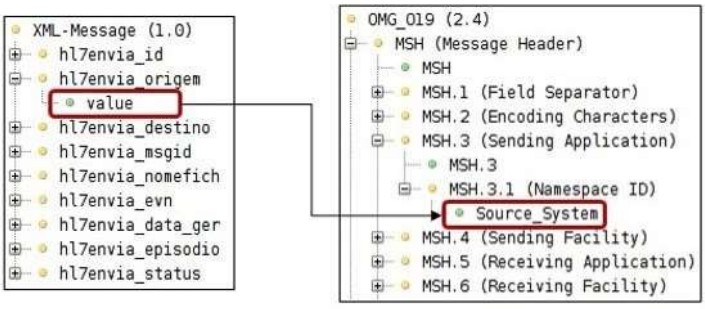


Figura 6.3: Mapeamento das variáveis para construção de mensagem HL7v2.

Tudo isto foi possível pela utilização das etapas do tipo *Message Builder*, através do mapeamento dos campos da mensagem de entrada para os campos da mensagem de saída. De seguida, de modo a registar o envio da mensagem, foi necessário registar o seu tempo. Para tal, foram mapeados os valores dos campos *Origem*, *Destino*, *MsgId* e *DataGer*, e também foi essencial a criação de uma tabela designada *hl7analisa* (tabela 6.2), onde são inseridos dados de análise.

1 https://github.com/avisi/jmeter-mllp-tcpclient

hl7analisa	
Atributo	**Tipo**
ID	int
Origem	varchar(45)
Destino	varchar(45)
MsgId	int
DataGer	timestamp
DataEnv	timestamp
DataAck	timestamp
IdAck	int
ErroHl7	int

Tabela 6.2: Caracterização da tabela hl7analisa.

Para a tabela acima foi essencial colocar o atributo *MsgId*, de forma a verificar a ordem pela qual as mensagens são enviadas, assim como a *DataGer*, para comparação entre o tempo em que a mensagem foi gerada e o tempo que a mensagem foi enviada (*DataEnv*). O campo *DataAck* retrata o tempo em que a mensagem demorou do *Mirth* para o sistema destino, e por último o *ErroHl7*, fornece a garantia se uma mensagem foi ou não recebida no sistema destino (valores 0 e 1, respetivamente). A inserção de todos estes dados foi realizada por uma etapa *JavaScript* do *source transformer* demonstrada no Apêndice A (excerto A.2), onde cada mensagem transformada corresponde a uma inserção na tabela *hl7analisa*.

Após a transformação da mensagem, é então enviada para a secção *Destination* deste canal. Nesta secção foram estipulados testes para um conetor do tipo *TCP*. Para a sua configuração, apenas foi necessário colocar o endereço *IP* e a respetiva porta, e o modo de transmissão escolhido foi *MLLP*.

Posteriormente, de modo a ser entendido quando a mensagem chega ao sistema destino, foi criado uma etapa *JavaScript* no *Response Transformer*. Tal como mencionado no sub-capítulo 4.2.3, este componente tem como principal objetivo de tratar os *ACK*'s das mensagens que chegam a um sistema recetor pretendido. Pelo *script* do excerto de código A.3, aquando a receção de uma mensagem de resposta são estabelecidas duas alternativas:

- Mensagem não foi recebida: Através da variável mapeada *msgid*, é atualizado o valor do atributo *ErroHl7* para 1 na tabela *hl7analisa*, indicando que a mensagem não foi entregue.
- Mensagem foi recebida: A conteúdo da mensagem *ACK* gerada pelo sistema externo destino, é redirecionada para o *ACK Channel*, que será abordado adiante.

Foi também utilizado outro conetor do tipo *Database Writer* de forma a testar diferentes tipos de conexões (excertos A.4 e A.5). Para tal foram usadas bases de dados do tipo *SQL* e *noSQL*, onde através das variáveis mapeadas com os valores da mensagem recebida, facilmente se fez o seu envio para a tabela/coleção *hl7recebe* do sistema externo pretendido.

Desta forma a tabela apenas contém atributos que são relevantes para a constituição de uma mensagem HL7 v2.

hl7recebe	
Atributo	**Tipo**
ID	int
Origem	varchar(45)
Destino	varchar(45)
Evn	varchar(45)
DataGer	timestamp
Episodio	int

Tabela 6.3: Caracterização da tabela hl7recebe.

6.2.2 *Destination Channel*

Depois de todos os componentes definidos anteriormente, este canal foi desenvolvido estrategicamente de modo a receber as mensagens do *Receiver Channel*. A sua principal funcionalidade é exercer o papel de um sistema externo destino, onde através de um conetor *TCP Listener*, com o modo de transmissão *MLLP*, são recebidas as mensagens.

Como abordado no sub-capítulo 4.2.2, existe uma opção designada *Source Settings*, que permite definir o momento de envio de uma resposta para o sistema de origem, quando é recebida uma mensagem. Assim, foi definida a opção *"Before Processing"*, ou seja, assim que a mensagem chega ao canal é enviada a resposta (figura 6.4).

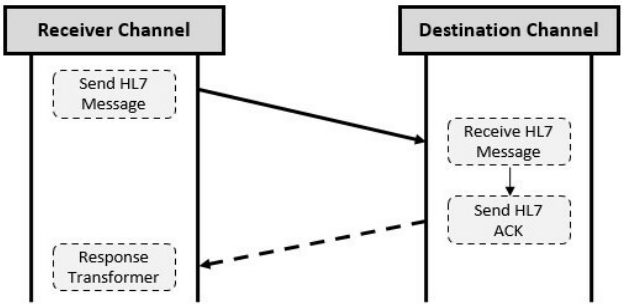


Figura 6.4: Geração de mensagens HL7 *ACK*.

Com este processo, é possível visualizar se uma mensagem foi recebida com sucesso no destino pretendido e, por outro lado, ter acesso ao momento em que dada mensagem foi recebida.

Neste canal, relativamente à secção *Destination* as mensagens foram encaminhadas para uma pasta local *recebe*, não sendo este passo muito relevante pois não constituí um elemento preponderante na análise de mensagens.

6.2.3 *ACK Channel*

Este canal tem como principal objetivo auxiliar o tratamento da resposta proveniente do sistema de destino. Desta forma, foi definido o conetor do tipo *Channel Reader* onde as mensagens são enviadas/recebidas internamente entre canais, sem protocolos. De seguida, foi estabelecido um filtro que faz a verificação da existência do campo *msgid*. Após esta validação, no *Source Transformer* os valores dos campos *msgid* (identificação da mensagem de original) e *ackid* (identificação da mensagem *ACK* gerada) são mapeados para variáveis internas do canal.

```
//Atribuicao das variaveis e insercao no arraylist
var ackID = $('ackID');
var msgID = $('msgID');
var params = new Packages.java.util.ArrayList();
params.add($('ackID'));
params.add($('msgID'));

//Definicao da query para atualizar a informacao de uma mensagem,
//conforme o seu id
var sql = "update hl7envia set IDACK = ?, DATAACK = sysdate(), ERROHL7 = 0
    where MSGID = ?";

//Conexao a base de dados e atualizacao dos dados da tabela hl7analisa
try {
  var dbConn = DatabaseConnectionFactory.createDatabaseConnection('
  com.mysql.cj.jdbc.Driver', 'jdbc:mysql://ipadress:port/dbteste',
  'root','password');
  dbConn.executeUpdate(sql, params);

} finally {
    if (dbConn) {
      dbConn.close();
  }
}
```

Excerto de código 6.5: Atualização da tabela *hl7analisa*.

Finalmente, na secção *Destination*, foi desenvolvido um *script* (excerto 6.5), de maneira a atualizar a restante informação da tabela *hl7analisa*. Como se pode observar, através das variáveis com os valores mapeados anteriormente no transformador, são atualizados os campos *idack* e *dataack*, relativamente á mensagem em questão. O valor deste último atributo foi atribuído por intermédio de um método da *API* do *Mirth* (*sysdate()*).

6.2.4 *Caminho crítico de uma mensagem*

A título de exemplo, pressuponha-se que uma mensagem tem como fonte uma base de dados *MySQL* e um destino *TCP*, idealizando o êxito no seu envio, percorrendo a arquitetura de teste anteriormente detalhada.

Inicialmente a mensagem é gerada através do *script* e os dados inseridos na tabela *hl7envia* numa base de dados *MySQL*. Posteriormente, através das sucessivas consultas do conetor de origem do *Receiver Channel*, os campos dos atributos são mapeados para variáveis internas do *Mirth*. De seguida, os dados são transformados numa mensagem HL7 v2 e, através dos valores mapeados da mensagem, são inseridos na tabela *hl7analisa*, para averiguar os tempos da mensagem. Após o transformador, o valor do atributo *Status* é modificado para 1 e a mensagem é enviada para a secção destino.

Nesta secção, a mensagem é enviada para o *Destination Channel*, e após a sua receção, retorna a validação para o canal de origem. Como foi enviada com sucesso, esta resposta é encaminhada para o *ACK Channel*, onde são mapeados para variáveis os valores do *id* da mensagem original e o *id* do próprio *ACK*. Por fim, através do *id* da mensagem original, na secção *Destination* são atualizados os valores dos atributos *idack* e *dataack*, na tabela *hl7analisa*.

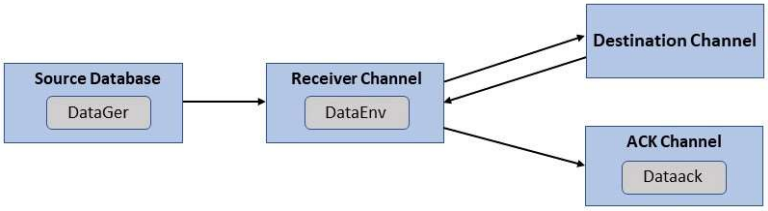


Figura 6.5: Caminho crítico de uma mensagem de acordo com a arquitetura de teste.

Como se pode constatar pela figura 6.5, pelo trajeto de uma mensagem, através da arquitetura de teste desenvolvida, é possível a monitorização dos tempos de determinada mensagem, desde que foi gerada até ao seu sistema destino.

6.3 Indicadores de Desempenho

Após o desenvolvimento de toda a infraestrutura e o estabelecimento de uma arquitetura de canais sólida e robusta na plataforma *Mirth Connect*, que proporcionasse uma monitorização adequada, segue-se a análise que foi efetuada a diferentes fluxos de mensagens.

No sub-capítulo 2.5.2 foram mencionadas regras que promovem o processamento de fluxo de dados ideal, que por sua vez também podem ser aplicadas a sistemas de *software* baseados em mensagens, ou seja, sistemas que contém implícita uma infraestrutura MOM

que é representativa por ser um dos elementos essenciais de arquiteturas ESB. Como se pode denotar, o objetivo de sistemas de processamento de fluxo é equivalente ao desta dissertação, ou seja, permitir a passagem de um elevado conjunto de dados/mensagens em tempo real.

Neste contexto, uma vez que a arquitetura ESB também se encontra incorporada no servidor do *Mirth*, toda a análise da plataforma foi realizada conforme algumas regras de processamento de fluxo e métricas apresentadas no estado da arte, no sub-capítulo 2.5.3. Desta forma é proporcionada a extração de indicadores de desempenho, que fazem variar o fluxo de mensagens que passam pelo *Mirth*.

Inicialmente foram realizados testes iniciais à plataforma, de forma a averiguar a capacidade que o *Mirth* apresenta para enviar e receber mensagens (tabela B.1). Estes testes foram realizados apenas com o transformador da origem ativo, para transformar os dados das bases de dados em mensagens HL7 e vice-versa. Quando foram analisadas diferentes fontes, o destino foi uma base de dados *MySQL*, tal como para a fonte, quando foram testados diferentes destinos.

Por uma questão de coerência de resultados, todos os testes realizados tem uma duração de aproximadamente 30 minutos. Pela apresentação do gráfico na figura 6.6, pode se verificar que o número de mensagens não ultrapassa as 24000 mensagens de forma substancial, pelo que através do *script* de geração de dados, fez sentido a inserção de 400 linhas de dados de 30 em 30 segundos. Desta forma, foram realizadas 60/62 inserções (excerto de código A.1), que perfaz aproximadamente as 24000 mensagens no total de 30 minutos. A mensagem HL7 v2, que foi introduzida como *template* é para eventos OMG_019, ou seja, o *trigger event* desta mensagem é para qualquer alteração ao nível da ordem de clínica geral. Cada mensagem contém um tamanho de 594 *bytes*.

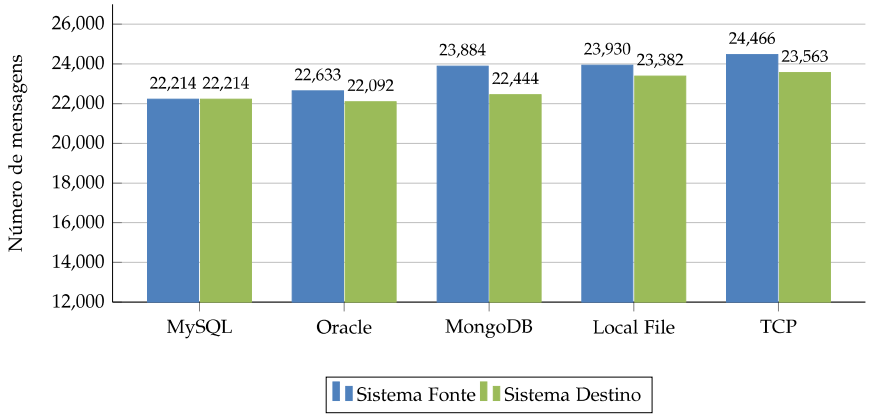


Figura 6.6: Resultados iniciais com sistemas fonte/destino diferentes, sem utilização dos componentes *Mirth*, cada teste com duração de 30 minutos.

Exceptuando os testes que geraram os resultados da figura acima, é de salientar que todos os testes foram efetuados com diferentes fontes e o mesmo tipo de sistema destino (*TCP*). Isto é importante para toda a análise, no sentido em que através deste tipo de sistema é possível a personalização da mensagem de confirmação (*ACK*) das mensagens que recebe, o que proporciona o acesso a todos os tempos do caminho crítico das mensagens (tempo de geração, tempo de envio e tempo de receção no sistema destino).

O ambiente de teste que permitiu a geração de todos os resultados, foi realizado numa máquina equipada com um processador Intel Core i7-8750H CPU 2.20 GHz, Memória *RAM* 15.5 GiB (DDR4) e Disco 190 GiB SSD NVme.

De seguida, são então representados todos os indicadores de desempenho extraídos ao longo de todos os testes efetuados sob a plataforma *Mirth Connect*. Todos gráficos apresentados, foram desenvolvidos conforme os resultados do número de mensagens apresentados nas tabelas dos apêndices B e C.

Tipo de Conetores de Origem

De acordo com os tipos de conetores da maioria das plataformas de integração analisadas no sub-capítulo 2.3.4, foram tidos em atenção 4 tipos de sistemas diferentes, nomeadamente bases de dados *SQL* e *noSQL*, *TCP* (*MLLP*) e ficheiros provenientes do *file system* da máquina local (consulta ou inserção de ficheiros de uma dada pasta da máquina onde o *Mirth* se encontra em execução).

Conforme os resultados demonstrados na figura 6.6, pode-se verificar que uns tipos de sistemas fontes apresentam um maior fluxo de mensagens em relação a outros, como é o caso dos tipos de conetores *readers* (*MySQL*, *Oracle*, *MongoDB* e *Local File*) e *listeners* (*TCP*). Em análise, esta situação pode representar um entrave quanto ao fluxo contínuo de mensagens, pela forma de como o *middleware* as recebe. Desta forma, torna-se essencial analisar o conetor de origem uma vez que o número de mensagens que chegam à plataforma depende apenas deste componente.

Relativamente aos conetores *listeners*, o tráfego normalmente é recebido a partir de protocolos (como *TCP* ou *HTTP*), exigindo que a conexão seja definida através de um endereço *IP* e de uma porta específica. Desta maneira, o motor de integração estará continuamente esperando mensagens de determinada fonte. Contrariamente dos conetores *readers*, que cujo tráfego é deliberado através da execução de uma *query* (por exemplo, consulta a uma base de dados ou consulta dos ficheiros contidos numa pasta local), num intervalo de tempo definido. Neste contexto, a pesquisa de dados/mensagens é realizada intermitentemente em períodos de tempo, definidos pelo utilizador.

Como expectável, os conetores *listeners* têm uma maior probabilidade de proporcionarem um maior fluxo de dados, devido à sua vantajosa caraterística de estar continuamente à

escuta através da rede. Observando o gráfico da figura 6.6 e a tabela B.1, existem algumas deduções que podem realizadas entre os sistemas utilizados:

- **Bases de dados *SQL***: entre os dois motores de armazenamento de dados deste tipo, pode-se denotar que a diferença de mensagens recebidas entre a base de dados *MySQL* e *Oracle* é de aproximadamente 400 mensagens. Contudo, este tipo de sistemas fonte apresenta um menor desempenho ao nível de consulta de dados, mesmo comparativamente a outros tipos de sistemas *readers*, como é o caso do *MongoDB* e *Local File*. Contrastando este tipo de sistema com os outros, o número de mensagens difere, aproximadamente, entre: 1100 mensagens com uma base de dados *MongoDB*, 1500 mensagens relativamente ao *Local File* e 2000 mensagens em relação a uma fonte *TCP*.

- **Base de dados *noSQL* (MongoDB)**: para um tipo de conetor *reader* este sistema apresenta bons resultados ao nível de consulta de dados. Isto advém do facto dos sistemas *noSQL* serem projetados para trabalhar com grandes quantidades de dados. Mesmo que a informação esteja devidamente estruturada, o ênfase deste tipo de bases de dados não se encontra na estrutura de dados, mas sim na forma como estes são armazenados, acedidos ou processados [79]. Assim sendo, torna-se vantajoso quando se trata de trabalhar com grandes volumes de dados em tempo real. Esta vantagem é denotada em comparação com a bases de dados *SQL* pelos resultados fornecidos.

- ***Local Files*:** relativamente aos tipos de conetores *readers*, apresenta-se como o modo de transmissão com maior taxa de mensagens recebidas no *Mirth*. Mesmo comparativamente ao sistema *TCP*, apenas difere entre cerca de 500 mensagens recebidas. Para este ambiente de teste, foi criada uma pasta local no *container*, preparada com 24000 mensagens HL7. Posteriormente, estas mensagens iam sendo retiradas conforme um *pooling* de 2 segundos. Estes resultados advém do facto da máquina onde estão a ser executados os testes, apresentar um disco *SSD*, que compreende caraterísticas como alto desempenho a nível da velocidade de leitura e gravação de dados. Contrariamente aos discos mecânicos (*HDD*), cujo desempenho diminui significativamente, e consequentemente poderia diminuir o fluxo de mensagens [80].

- ***TCP* (*MLLP*):** tipo de transmissão que exibe maior eficiência relativamente aos restantes tipos sistemas demonstrados, por ser um conetor do tipo *listener*. Como já referenciado na explicação dos conetores de origem utilizados do *receiver channel* (página 69), os dados são geradas consecutivamente pela ferramenta *JMeter*, com um número definido de 2 *threads* a enviar mensagens para o *Mirth*.

Manter os dados em movimento

Relativamente à quantidade de informação que é recebida a partir de determinado sistema fonte, em muitos dos casos, o problema pode não estar relacionado com a taxa de transmissão para o sistema externo, mas sim do modo de como o motor de integração recebe os dados e a quantidade de componentes internos utilizados para os tratar. Esta questão pode gerar algum atraso no que diz respeito à receção da informação.

De forma a diminuir esta latência em relação à aquisição e processamento de dados, o sistema deve ser capaz de realizar o processamento de mensagens sem ter operações intensivas de armazenamento no caminho de processamento crítico. Deste modo, o sistema deverá ser capaz de realizar o processamento de mensagens "*in-stream*", sem ter associadas operações de armazenamento, que podem causar atraso relativamente ao fluxo. Conforme o explicado, essa mesma lógica foi testada no *Mirth* (figura 6.7), em que os ficheiros de *logs* e mensagens no seu formato *raw* são guardados numa bases de dados *MySQL*.

Para tal, foram realizados testes conforme os diferentes modos de armazenamento que a plataforma de integração permite, sendo estes já descritos no sub-capítulo 4.2.1.

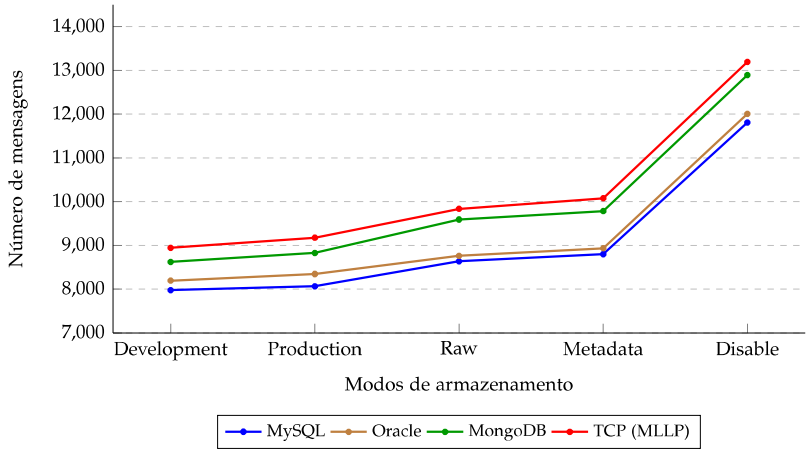


Figura 6.7: Resultados com diferentes modos de armazenamento do *Mirth*.

Em análise da figura 6.7, o modo *Development* é o que apresenta um valor mais baixo de mensagens pois guarda tudo acerca da mensagem. Este modo possibilita a disponibilização de informação ao utilizador do histórico de mensagens. Tanto como para as mensagens recebidas na secção de origem do canal como para mensagens da secção destino, o armazenamento é efetuado para as mensagens em bruto, transformadas, codificadas, mensagens *ACK* recebidas (para todas as mensagens recebidas e enviadas) e meta-dados da mensagem. Analisando o modo *Production*, a diferença não é significativa, pois em relação ao anterior,

apenas não guarda informação relativamente às mensagens processadas na secção de origem do canal.

Contrariamente ao modo *Raw*, apresenta um aumento de cerca de 1000 mensagens, apenas armazenando as mensagens recebidas/enviadas no seu formato em bruto e meta-dados. O modo *Metadata* não demonstra grande diferença do anterior. Tal como o seu nome indica, guarda informação relativamente aos meta-dados de todas as mensagens, como data e hora de receção, envio e confirmações de envios, erros, estado das mensagens, etc. Por último, o modo *Disable* apresenta a melhor performance com, aproximadamente, mais 3000 mensagens que o anterior. Este tipo de armazenamento não armazena qualquer tipo de informação informação relativamente às mensagens, apenas o número de mensagens recebidas, enviadas, filtradas ou em fila.

Pela a análise efetuada, pode-se constatar que o indicador, em termos de armazenamento, que mais influência o fluxo de mensagens é a gravação dos meta-dados, apresentando uma diferença de cerca de 3000 mensagens em relação ao armazenamento de dados em bruto.

De seguida, é demonstrado um gráfico, na figura 6.8, que contém os resultados obtidos de testes realizados de mensagens adquiridas sem/com o auxílio do componente de filtragem do *Mirth*, apenas para base de dados *SQL*.

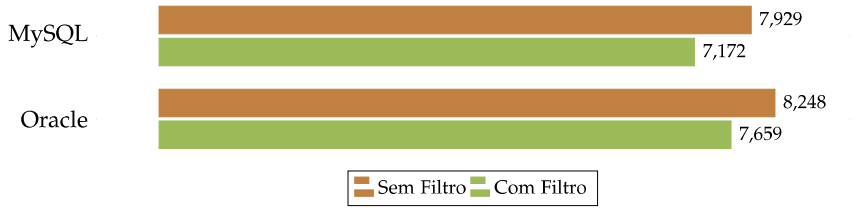


Figura 6.8: Número de mensagens recebidas, com/sem o componente de filtragem ativo.

Como se pode reparar pela figura acima, o fluxo de mensagens não é apenas afetado pelo armazenamento da informação que o *Mirth* recebe, mas também por toda a atividade que realize processamento com os dados, isto é, componentes internos da ferramenta de integração que desempenham determinadas ações com os dados recebidos. Neste caso, o componente de análise é o de filtragem, que através de regras define as mensagens que podem prosseguir para o estado de transformação para serem processadas.

Para o processo de filtragem, foi mapeado o campo *status* da mensagem, através de um template *inbound* definido, e os dados apenas são aceites se contiver o valor o (para não processar dados repetidos, explicado no sub-capítulo anterior). Caso contrário, as mensagens são filtradas e não seguem para a fase de transformação.

Sem o componente de filtragem do *Mirth*, a consulta à base de dados é realizada diretamente através da *query*, com clausula *"WHERE status=o"*, já mencionado em 6.2.1.

Através desta comparação demonstrada na figura 6.8, com a aplicação do filtro, apresenta um decréscimo de aproximadamente 700 mensagens, ou seja, representa um bloqueio no fluxo de dados.

Um outro termo de relação foi a comparação entre os resultados iniciais (tabela B.1) e os resultados com a aplicação da arquitetura de teste (tabela B.4 - 1 *thread*). Como já referido, os testes para obter os resultados iniciais apenas foram usadas etapas do *source transformer* para transformar os dados em mensagens, sem qualquer utilização da etapa *javascript source transformer* ou *destination response*, ao invés dos testes realizados com a arquitetura de canais desenvolvida.

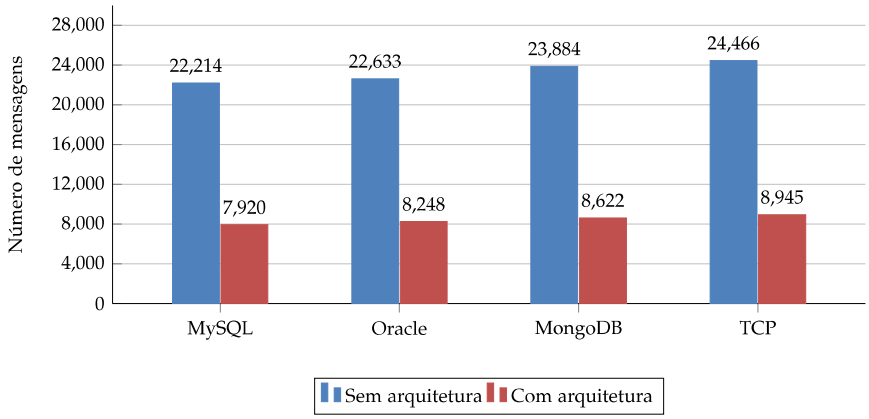


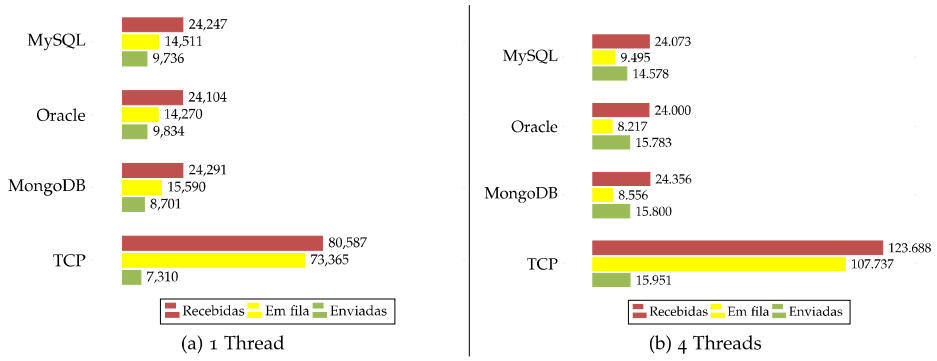
Figura 6.9: Resultados sem/com utilização da arquitetura de canais desenvolvida.

Os resultados desta comparação são explícitos no gráfico de barras da figura 6.9. Em análise, a diferença é significativa para qualquer tipo de fonte apresentada, evidenciando uma redução de aproximadamente 14000 mensagens, com o uso da arquitetura de canais. Esta quebra é explicada pelas atividades internas do *Mirth*, executadas durante o processamento de dados, onde no componente *source transformer* cada mensagem é inserida numa base de dados *MySQL* e, posteriormente, no *destination response* a mensagem de confirmação (HL7 *ACK*) recebida do sistema externo destino ser reencaminhada para um outro canal (*ACK Channel*).

Ativação das opções de filas e *multithreading*

Como já referido no sub-capítulo 2.5.1, uma fila de mensagens é uma técnica utilizada para a comunicação entre sistemas, através de uma interface para permitir a passagem de mensagens. Normalmente, esta abordagem é aplicada quando a velocidade de aquisição e processamento de mensagens é menor do que a velocidade da taxa de envio de um outro

sistema. Assim, o processo de filas de mensagens é caraterizado pelo seu modo assíncrono em relação a sistemas de envio, isto é, as mensagens são inseridas num *buffer* possibilitando aos sistemas de envio a transmissão contínua de mensagens, que se vão acumulando. Para tal, foram realizados testes para verificar a incidência desta técnica (resultados na figura 6.10a), sendo definido um tamanho limite do *buffer* de 1000 mensagens. Isto não significa um limite quanto ao número permitido de mensagens na fila, ou seja, pode-se enfileirar milhões de mensagens e todas elas residem na base de dados ou no disco. No entanto, o número máximo que permanece na memória, enquanto a fila está acumulando mensagens, é definido pelo tamanho do *buffer* da fila.



(a) 1 Thread (b) 4 Threads

Figura 6.10: Resultados com filas ativas com utilização de 1 e 4 threads, respetivamente.

Pela observação do gráfico 6.10a, este método inviabiliza a acumulação de uma elevada quantidade de mensagens que proporciona um aumento do consumo de memória da máquina, prejudicando no processamento das mesmas e no envio. Em relação ao sistema fonte do tipo *TCP*, torna-se claro este caso de estudo, em que o *Mirth* vai acumulando consecutivamente as mensagens chegadas do sistema fonte, afetando posteriormente o envio das mesmas. É de salientar que o algoritmo de ordem da fila exercido é *First In First Out (FIFO)*, ou seja, a primeira mensagem a entrar na fila será a primeira a ser processada.

Perante os resultados obtidos, tornou-se necessário a adoção de mecanismos que vão retirando as mensagens da fila a uma taxa maior. Isto deve-se ao facto dos dados estarem a ser retirados da fila de forma síncrona pois só está a ser usada 1 *thread* para processamento, ou seja, uma mensagem é processada e só depois de ser enviada é que processa a seguinte. Deste modo, a solução empregue foi a utilização de mecanismos *multithreading*. Desta forma, as mensagens que se encontram na fila vão ser processadas paralelamente, ao invés de serem processadas uma única de cada vez.

Pela observação da figura 6.10b, verifica-se uma melhoria bastante acentuada principalmente em relação às bases de dados. Em análise do gráfico com a aplicação de 4 *threads* para

processamento dos dados, o número de mensagens recebidas mantém-se igual, diminuindo a carga implícita na fila de mensagens notoriamente, o que consequentemente aumenta o número de mensagens enviadas. Em relação à fonte *TCP*, o número de mensagens recebidas e em fila aumentou devido ao envio contínuo e simultâneo de mensagens de dois clientes virtuais (2 *threads*), por parte da ferramenta *JMeter*.

Na tabela 6.4, são abordados os tempos de geração e de envio com uma fonte *MySQL*, conforme mensagens inseridas num intervalo relacionado com as mensagens processadas do *Mirth*. Os testes foram realizados para 1 e 4 *threads*, de forma a possibilitar a comparação entre os dois métodos.

	1 Thread		
Id de Mensagens	**Tempo de Geração**	**Tempo de Envio**	Δ_t
1	0m0s	0m0s	0m0s
2500	3m01s	6m56s	3m55s
5500	6m34s	15m11s	8m37s
7500	9m04s	20m55s	11m51s
9500	11m35s	26m45s	15m10s
	4 Threads		
1	0m0s	0m0s	0m0s
3500	4m04s	5m54s	1m50s
7500	9m04s	13m0s	3m56s
10500	13m02s	18m35s	5m33s
14500	18m02s	26m06s	8m04s

Tabela 6.4: Resultados dos tempos conforme intervalos de mensagens, com filas ativas e sem/com *multithreading*.

De forma a clarificar os parâmetros da tabela 6.4, o tempo de geração e de envio estão representados de forma crescente, tendo como base o tempo inicial da primeira mensagem. A variável Δ_t simboliza a diferença entre os dois tempos, ou seja, o tempo que a mensagem demorou desde que foi gerada até ser enviada do *Mirth* para o sistema destino.

Em análise, pelos tempos obtidos com uma única *thread*, a diferença dos tempos acentua-se cada vez mais à medida que as mensagens são geradas. Este *bottleneck* propaga-se devido à acumulação contínua de mensagens na fila, pois existe apenas 1 *thread* a processar todas as mensagens, isoladamente. Relativamente aos resultados com 4 *threads*, os valores da diferença dos tempos apresentados, comparativamente com os de 1 *thread*, são significativamente menores pois as mensagens não se encontram tanto tempo na fila devido à opção ativa de *multithreading*. Este balanço constituí um outro comprovativo do facto deste indicador representar um benefício quanto ao fluxo constante de dados de uma ferramenta de integração.

No entanto, pode constituir um problema caso exista o requisito de manter a ordem de envio das mensagens. Isto é explicado pelo facto das mensagens serem processadas

paralelamente por diferentes *threads*, por exemplo, uma mensagem de ordem 4 pode ser processada mais rapidamente do que uma mensagem de ordem 2, onde o envio da 4 será antes da mensagem 2.

Também foram realizados testes com 4 *threads* e sem filas ativas (em anexo: tabela B.4), mas o fluxo apresentado para as bases de dados foram idênticos ao da utilização com 1 *thread*. Estes resultados são explicados pela razão das consultas efetuadas às tecnologias de armazenamento de dados constituírem um processo de *thread* única. Já para a fonte *TCP* o fluxo aumentou pois na ferramenta *JMeter* estão definidos 2 clientes e enviar constantemente mensagens para o *Channel Receiver*, o que aumentar as opções de configuração para 4 *threads* de processamento beneficia o número de mensagens.

Particionamento e Balanceamento da carga

Com o intuito de promover um maior fluxo de dados quanto às mensagens enviadas, uma outra solução analisada foi a distribuição das mensagens provenientes do sistema fonte por mais do que um destino. Desta forma a implementação deste método seria no *Receiver Channel*, onde as mensagens seriam enviadas da sua secção *source* para 2 destinos (dois *Destination Channels*), de forma a paralelizar a carga de mensagens recebidas. No entanto, este tipo de abordagem não funciona da maneira prevista no *Mirth*, pois o seu modo de transmissão de mensagens é síncrono, como se pode observar no exemplo da figura 6.11.

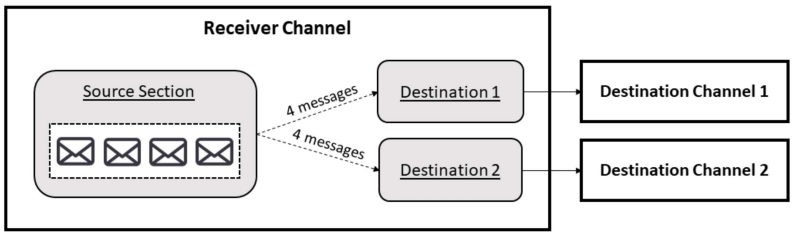


Figura 6.11: Exemplo do modo de transmissão síncrono de um canal.

O exemplo demonstrado na figura 6.11 é representativo da forma de como as mensagens fluem num canal do *Mirth*. Deste modo, as mensagens chegam ao canal e são processadas no *source transformer*, sendo posteriormente encaminhadas para a secção destino. Nesta secção, cada mensagem é enviada consoante os destinos estabelecidos, não havendo modo de as distribuir de forma a balancear a carga de mensagens, ou seja, cada mensagem é processada desde que é recebida até ser enviada. Uma solução a esta adversidade seria a utilização de uma etapa "*Destination Set Filter*", onde é decidido o destino de cada mensagem. Mas, como já foi apresentado no segundo indicador de desempenho, este método iria introduzir ainda mais processamento a cada mensagem, o que se refletiria no fluxo.

Visto o problema do modo de transmissão e o da sobrecarga de um canal com filas ativas aquando a rececão contínua de dados, a solução passou por criar um outro *Channel Receiver* com o objetivo principal de aumentar o fluxo de dados, equilibrar a carga e, consequentemente, aumentar o número de mensagens recebidas/enviadas. Desta forma, foram realizados testes nesse âmbito, onde os dois canais recetores do *Mirth* fazem consultas a uma mesma fonte de dados (no caso de bases de dados), ou os escutam para uma mesma porta de rede (via *TCP*). Na figura 6.12, é demonstrada a comparação dos resultados com 1 e 2 *Receivers Channels* com *multithreading* (4 *threads*) e filas ativas, tendo em conta o número total de mensagens enviadas do *Mirth*, sendo estas relevantes porque representam as mensagens já processadas.

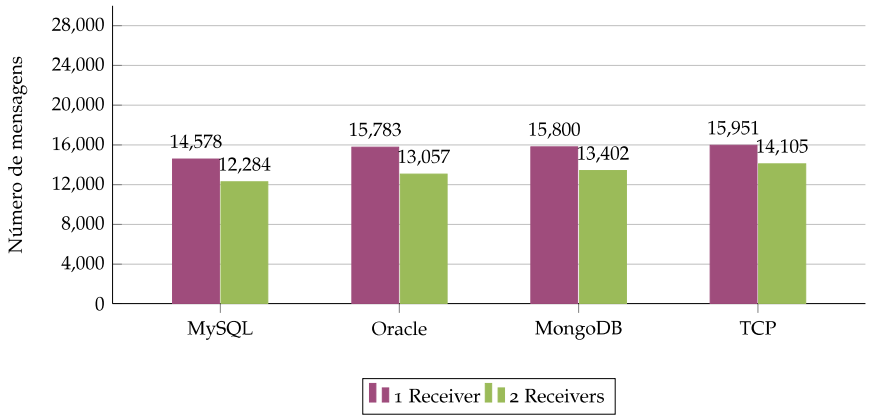


Figura 6.12: Comparação dos resultados do número de mensagens enviadas de 1 e 2 *Channels Receiver*, com filas ativas e 4 *threads*.

Pela análise da figura 6.12, denota-se um decréscimo significativo do número de mensagens com a adição de um outro canal recetor. Conforme as tabelas B.5 e B.6 apresentadas no apêndice B, estes resultados exibem uma diminuição dos dados processados mas um aumento bastante acentuado relativamente à quantidade de mensagens acumuladas na fila. Esta sobrecarga é proveniente do número de operações que estão a ser realizadas simultaneamente no *Mirth* causando uma quebra na sua performance. Este tipo de operações estão associadas às consultas e inserções de informação em bases de dados dos dois *receivers*, transformações dos dados em mensagens HL7 bem como o seu mapeamento para variáveis e armazenamento de dados para efeitos de histórico de mensagens. Assim, a realização de todas estas atividades em simultâneo dos dois canais recetores gera um grande atraso no processamento de mensagens.

Ao longo dos testes realizados, com o auxílio do comando *docker stats*, foi realizada a monitorização da utilização dos recursos utilizados do *container*, onde o *Mirth* se encontrava

em execução. Como tal, foram feitas medições do uso de CPU, de memória *RAM*, operações de escrita/leitura no *file system* do *container* (*Block I/O*) e do tráfego de rede (*Network I/O*). Estes dois últimos recursos apresentavam valores cumulativos durante as execuções dos testes, pelo que a cada novo teste era necessário reiniciar o *container* para os valores serem credíveis. Todos os resultados obtidos encontram-se no apêndice C.

Pela análise dos resultados desta monitorização pode observar-se, pela tabela C.2 (apêndice C), que os valores do CPU já eram significativos apenas com 1 *Channel Receiver*, apresentando uma utilização na ordem dos 50%. No entanto, pela tabela C.4, com a adição de um outro *receiver* esses valores acentuaram-se para a ordem de 75% de utilização. Pela apreciação deste estudo, tal significa que a carga de trabalho exercida numa instância do *Mirth* era demasiado elevada, afetando a sua performance quanto ao fluxo de mensagens.

Relembrando o método de balanceamento da carga de mensagens e tendo em conta o uso elevado do *CPU* com apenas uma instância, a solução passou pelo particionamento do *Mirth* através da criação de várias instâncias, ou seja, uma instância *Mirth* por *container*. Assim foram criadas 3 instâncias da plataforma de integração em estudo: duas para os dois *channels receivers* (e os respetivos ACK *channels*) e uma outra com o *channel destination*. Desta forma, é feito o balanceamento da carga, tanto a nível do fluxo de mensagens como a nível de utilização dos recursos, de forma a promover o bom desempenho do *Mirth*.

Na figura 6.13, estão representados os resultados de uso de CPU, de forma a facilitar a comparação entre os 3 ambientes de teste: com 1 instância com 1 *Receiver Channel*, 1 instância com 2 *Receiver Channels* e com 2 Instâncias com 2 *Receiver Channels*. Para este último caso de teste, foi efetuada a média de percentagem de CPU dos dois *receivers*, com as diferentes fontes. A Instância 3 das tabelas C.5 e C.6 representam o *destination channel*. É de evidenciar que estes testes foram realizados com filas ativas e uso de 4 *threads*.

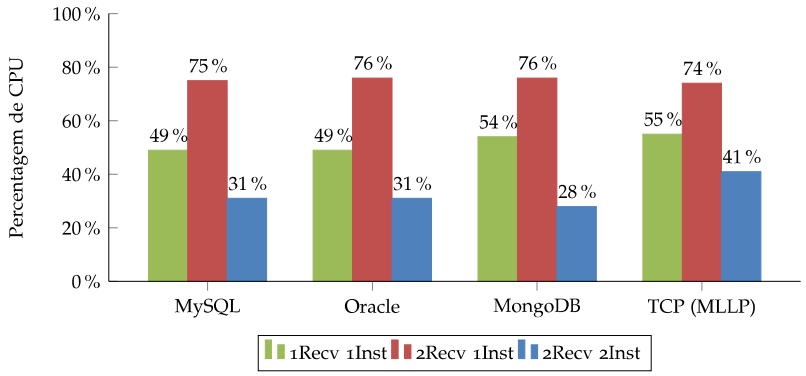
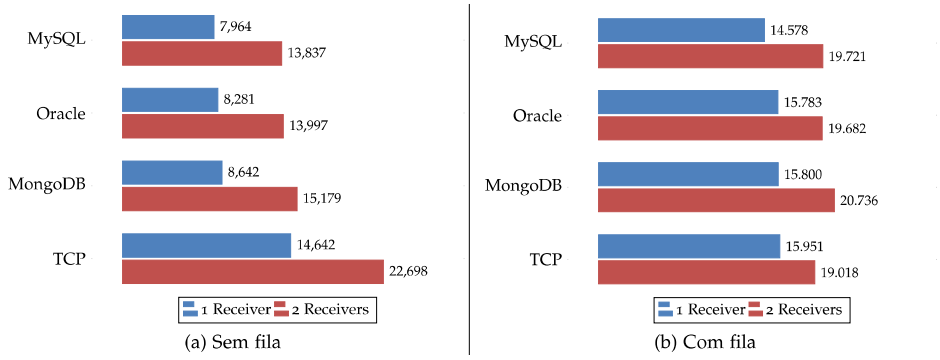


Figura 6.13: Utilização de CPU para 1 e 2 instâncias, com 1 e 2 *Receivers Channels*.

Pela análise dos resultados, a instanciação do *Mirth* em vários *containers* baixou notoriamente o uso de CPU, devido à divisão de trabalho exercido pela plataforma. Para as fontes do tipo *database reader*, a diminuição é maior do para fontes do tipo *TCP Listener* devido à taxa de envio desta última ser bastante maior. Isto resulta numa maior acumulação de mensagens nas filas, sobrecarregando um pouco mais o sistema de *software*. Como é obvio, este método e esta diminuição de uso de CPU, reflete-se no fluxo de mensagens apresentado no gráfico da figura 6.14.



(a) Sem fila (b) Com fila

Figura 6.14: Comparação de resultados do número de mensagens enviadas com 1 instância (1 *Receiver Channel*) e 2 instâncias (2 *Receivers Channels*), com 4 *threads* ativas.

Os resultados expostos no gráfico 6.14, revelam o bom desempenho de balancear toda a carga a partir da instanciação do *Mirth*. A comparação realizada, apresenta os resultados dos testes efetuados com uma instância e um *receiver channel* e com duas instâncias e dois *receiver channels*.

Pela análise dos gráficos 6.14a e 6.14b, o fluxo de mensagens inerente aos testes com 2 *receivers* demonstra a sua eficácia, no entanto, existe uma notória diferença de mensagens entre os dois ambientes de testes sem e com fila para a fonte *TCP*. Tal como já foi explicado anteriormente, isto deve-se ao facto do envio contínuo de grandes volumes da ferramenta *JMeter*, onde as mensagens vão se acumulando em demasia na fila e prejudica o fluxo. Visto isto, o ambiente de teste com duas instanciações do *Mirth* a receber mensagens através de uma fonte *TCP*, sem filas ativas e com *multithreading* representa o caso com maior número de mensagens enviadas.

Em relação aos restantes recursos utilizados da máquina onde foram realizados os testes, averiguando os resultados obtidos no apêndice C pode-se afirmar que:

- A memória *RAM* é mais utilizada pelas fontes do tipo *database reader* do que do tipo *TCP listener*, que advém do facto do sistema estar constantemente a efetuar consultas às base de dados enquanto que para o *listener* apenas fica à escuta numa porta de

rede, exigindo menos esforço computacional. Para os testes efetuados com filas ativas, a memória aumenta pois as mensagens acumulam-se na fila, e em consequência, sobrecarrega a memória e realiza mais inserções na base de dados do *Mirth*.

- Relativamente à *network*, a fonte do tipo *TCP* apresenta uma maior taxa de dados lidos devido à quantidade de mensagens recebidas ser relativamente maior aos das outras fontes. Consequentemente, a quantidade de *megabytes* enviados será também maior (para o *destination channel*). Denota-se um decréscimo do tráfego de rede para os testes efetuados sem filas ativas, pois a taxa de aquisição de dados torna-se menor.
- Em relação à escrita/leitura no *file system* não foram obtidos valores significativos que possam alterar o desempenho do *Mirth*.

6.4 AVALIAÇÃO DOS RESULTADOS

Com base nos resultados obtidos, ao longo da introdução e explicação das métricas pode-se reconhecer e constatar algumas medidas que podem ser estipuladas de modo a melhorar o desempenho de plataformas de integração, proporcionando um bom fluxo de mensagens conforme diferentes cargas a que possa estar sujeito.

Relativamente ao primeiro indicador (tipo de conetores de origem), são analisados os diferentes tipos de sistemas fontes, com o objetivo de averiguar os que representam maior velocidade no diz respeito à sua taxa de envio e capacidade de taxa de aquisição do *Mirth*. Desta forma, pelos tipos de sistemas *readers*, o sistema de bases de dados *noSQL* apresentou melhores resultados, em relação ao tipo *SQL*, devido à sua elevada capacidade de trabalhar com volumes de dados em grandes escalas. Comparando o *MongoDB* com a leitura/gravação no *file system*, este último apresenta aproximadamente o mesmo fluxo, todavia pode não constar de um bom método relativamente à leitura/escrita de informação, se a plataforma estiver sujeita a grandes cargas de dados durante períodos longos de tempo. Como consequência, pode sobrecarregar em demasia o disco, o que afeta quanto à sua celeridade de taxa de gravação e leitura, se a taxa de aquisição do motor de integração não for suficiente.

Averiguando o tipo de conetor *listener*, a plataforma ao invés de estar a executar continuamente consultas a bases de dados ou *file system*, apenas fica à escuta numa determinada porta de rede. Pelos resultados da fonte *TCP*, constitui o melhor fluxo de mensagens testados no sistema de *software*.

Tendo em consideração a segunda métrica (manter os dados em movimento), observa-se a melhoria da performance da ferramenta de integração através da diminuição de utilização de componentes, que possam interferir no caminho de crítico de cada mensagem. Neste contexto, as operações de armazenamento de dados acerca das mensagens, para apresentação

na interface do *Mirth*, apresentam uma grande quebra no caminho das mensagens, tendo uma acrescida incidência no fluxo. O tipo de armazenamento que causava maior impacto era o de gravação de meta-dados, que são dados relativos a datas de receção, envio e de confirmação, possíveis erros, etc, sendo o melhor modo de armazenamento o *disable* (gráfico 6.7).

Pela comparação dos resultados sem e com arquitetura de teste, demonstra uma diferença de aproximadamente 14000 mensagens (gráfico 6.9), pela quantidade de componentes internos ativos, assim como o número de operações de inserções e *updates* realizadas na base de dados.

Observando o terceiro indicador (ativação das opções de filas e *multithreading*), foi usada uma abordagem relacionada com filas em *middlewares* devido à sua caraterística de assincronismo entre o *Mirth* e o sistema fonte, aumentando o número de mensagens recebidas. A utilização de filas de mensagens é também benéfica no sentido em que caso exista uma falha do sistema, as mensagens recebidas estarão temporariamente armazenadas numa base de dados ou no disco, o que significa que nunca serão perdidas. Este método representa um bom desempenho quando a taxa de envio da plataforma de integração apresenta uma velocidade relativamente adequada para a taxa de receção. Caso não se verifique, gera uma sobrecarga de mensagens no *Mirth* que prejudica o seu desempenho. De forma a solucionar este *bottleneck*, o uso de *multithreading* acelera o processamento de mensagens da fila, promovendo uma maior taxa de envio.

Para o último indicador de desempenho (particionamento e balanceamento da carga), o objetivo implícito é divisão dos *Receivers Channels* de modo a aumentar o taxa de envio de mensagens provenientes de um sistema fonte. Inicialmente este método foi aplicado para apenas uma instância do *Mirth*, que apresentou resultados significativamente piores devido ao esforço computacional exigido, mais especificamente em relação ao CPU. Com a instanciação do *Mirth* em três *containers* (2 *receivers* e um *destination*), a carga dos recursos utilizados pelo *software* é balanceada (gráfico 6.13), aumentando o fluxo de mensagens (gráfico 6.14).

Para cada métrica mencionada, denota-se explicitamente a melhoria em relação à taxa de mensagens recebidas e enviadas, no entanto, este bom desempenho é obtido conforme alguns aspetos que se pode ter em consideração. Assim, serão apresentadas algumas destas condições a ter em vista, de modo a atingir a melhor estrutura lógica consoante alguns possíveis casos de uso:

- **Uso de conetores *readers* e *listeners*:** Por norma, as plataformas de integração são constituídas por conetores já definidos, que apenas basta introduzir, pelo *driver*, a base de dados a utilizar. No entanto, o *Mirth* possui a possibilidade do desenvolvimento de código através de *JavaScript*. Esta linguagem possui imensas bibliotecas que inviabilizam a conexão de diferentes tipos de bases de dados. Através de modelos de

código e *scripts* globais, caso o utilizador utilize mais do que uma vez a mesma base de dados, poderá utilizar o mesmo *script*, ao invés de estar a criar novas conexões, evitando código duplicado e utilização de componentes internos da plataforma. Ainda relativamente a este tipo de conetor, pelo desenvolvimento de código para permitir a conexão a este tipo de sistemas externos, pelos resultados demonstrados anteriormente, a execução de *scripts* apresenta um menor esforço computacional do que o uso dos conetores já integrados no próprio motor de integração. Relativamente ao tipo de conetor *tcp*, apresenta ser um tipo de comunicação eficaz para elevados volumes de mensagens, como já evidenciado várias vezes anteriormente.

- **Utilização de componentes já integrados na plataforma de integração:** Ao longo da passagem de mensagens nos canais do motor de integração, estas são armazenadas na bases de dados de *logs* do *Mirth*, para os dados relativos a cada mensagem serem disponibilizados ao utilizador numa interface. Este processo inclui várias inserções dos dados em bruto e atualizações de estado, como datas de receção e envio de dados, que são armazenadas no motor de armazenamento de dados. Uma alternativa a este método seria desabilitar esta opção, modo de armazenamento *Development* para *Disable*, e usar um tipo de armazenamento idêntico ao utilizado na arquitetura de canais, apresentado anteriormente. Desta forma, apenas se registava os tempos pretendidos numa etapa do transformador, representando apenas uma inserção em determinada base de dados por cada mensagem. Relativamente aos componentes que fazem parte do processamento dos dados, devem ser utilizados o mínimo possível, de forma a não representar mais uma "paragem"no caminho crítico de cada mensagem durante o seu processamento.

- **Uso de filas de mensagens e *multithreading*:** Como já mencionado, esta abordagem permite uma maior aquisição de dados provenientes de um sistema fonte externo. Deste modo, quando a taxa de envio do sistema fonte é maior do que a taxa de receção do sistema de integração, este método é fundamental. No entanto, se a velocidade de processamento das mensagens alocadas na fila não for suficiente, o sistema pode sobrecarregar devido à elevada quantidade de dados acumulados na fila. Para tal, através de várias *threads* a executarem em simultâneo, este *bottleneck* é atenuado. Todavia, caso as filas não estejam habilitadas, o *multithreading* apenas é viável para sistemas fontes que enviem mensagens em paralelo, por exemplo, várias fontes *TCP* a enviarem mensagens para uma mesma porta. Caso contrário, aumentar o número máximo de *threads* de processamento não ajudará necessariamente, pois o cliente não utiliza uma abordagem de envio paralela.

- **Replicação de canais:** Por vezes, existem vários canais a serem executados em simultâneo, na expectativa de aumentar o processamento de mensagens para uma

mesma fonte. Esses mesmos canais podem ainda ter associados vários componentes para transformar ou processar os dados. Esta duplicação de trabalho, devido à atividade implícita em cada canal, pode causar uma elevada utilização de recursos da máquina, o que torna o sistema de integração sensível e, consequentemente, prejudica a sua performance quanto ao fluxo de mensagens. Com isto, para não exigir tanto esforço computacional, a instanciação é algo crucial pois não são partilhados os mesmos recursos, balanceando a carga.

7

CONCLUSÃO E TRABALHO FUTURO

O tipo do sistema de *software* analisado neste projeto, insere-se numa importante temática da atualidade, a interoperabilidade. Na área da saúde, com a elevada importância da informação clínica, existe a necessidade de distribuição da informação entre os diversos serviços e especialidades, sendo fulcral a comunicação entre os sistemas de informação. De forma a promover esta partilha de informação, independentemente da sua heterogeneidade, torna-se fulcral a aplicação de plataformas de integração.

Neste contexto, o trabalho desenvolvido no âmbito desta dissertação tem como principal objetivo a análise de interfaces HL7, mais especificamente sistemas de *software* de integração, de forma a efetuar a aquisição de métricas que promovem o seu desempenho a nível do fluxo de mensagens. Em termos da aplicabilidade destes indicadores num contexto real, é vantajoso ter em consideração as circunstâncias em que este tipo de plataformas operam em condições ideais, pois a sua execução sem qualquer tipo de sobrecarga no fluxo de mensagens é algo fundamental. Tudo isto devido ao facto deste tipo de ferramentas estarem integradas em instituições de saúde que auxiliam os prestadores de assistência médica e, qualquer tipo de paragem ou atraso na troca de dados entre sistemas, poderia ter graves consequências.

A primeira fase desta dissertação é dedicada ao estado da arte, onde foi analisado e estudado o conhecimento académico inerente ao tema proposto nesta investigação. Foram analisadas diferentes plataformas de integração na área da saúde para posterior análise, no entanto, a maioria apenas apresentavam versões de teste com tempo de utilização limitado. A única *open-source* que englobava todas as características e funcionalidades da maior parte dos motores de integração, era o *middleware Mirth Connect* (ou *NextGen Connect*), sendo o objeto de estudo deste trabalho. Conforme as infraestruturas MOM implícitas neste tipo de plataformas, foram averiguadas abordagens que promovem a sua performance, como as filas de mensagens e processamento de fluxo de dados.

Depois da análise realizada dos conceitos essenciais a este projeto, durante o capítulo 3 é exposta a metodologia de investigação que sustenta o processo de aplicação dos conceitos anteriormente pesquisados. Para tal, no capítulo 4 retratou-se a arquitetura e funcionalidades

do *Mirth* detalhadamente, já que este é o ponto primordial desta investigação. No capítulo seguinte são apresentadas as tecnologias essenciais que auxiliaram todo trabalho realizado.

Deste modo, tendo por base todo o conhecimento recolhido, para monitorização do fluxo de mensagens foi necessário a criação de uma arquitetura de canais que permitisse o acompanhamento de cada mensagem, de forma a escrutinar os pontos de falha durante o seu caminho. Após a realização de vários ambientes de teste com diferentes configurações e sistemas externos, foram registados indicadores que evidenciavam notoriamente o aumento do desempenho do *Mirth*, sendo posteriormente analisados com maior detalhe e descritos neste trabalho.

Nos objetivos da presente dissertação foram colocadas questões de investigação, que foram abordadas ao longo de todo o estudo efetuado. Neste contexto, de forma a concluir esta investigação e de modo a relacionar todos os conceitos e métricas apresentadas, as questões de investigação serão respondidas seguidamente.

Questão 1: Quais os indicadores que retratam uma boa performance para a plataforma de integração em estudo?

Depois da análise efetuada acerca das funcionalidades e componentes do *Mirth*, bem como os conceitos retidos acerca das boas práticas para um proveitoso processamento de fluxo de mensagens, foi possível ter em conta várias métricas que promovem o bom desempenho de motores de integração.

O primeiro indicador relaciona-se com o tipo de conetor fonte a ser utilizado. Desta forma, por norma destacam-se dois tipos de conetores: os *readers* e os *listeners*. O uso de conetores *readers* implica a consulta de dados efetuada conforme um período de tempo estabelecido pelo utilizador, o que pode gerar algum atraso quanto à rececão de informação. Já relativamente aos *listeners*, a plataforma fica à escuta numa determinada porta de rede, e o tráfego de dados apenas fica dependente da velocidade a que o sistema externo envia os dados.

O segundo indicador retrata a utilização de componentes inerentes ao motor de integração. Habitualmente, o modo de processamento dos sistemas de integração é realizado de forma síncrona, ou seja, processam uma mensagem de cada vez. Desta forma, quanto mais processamento cada mensagem contrair no seu caminho crítico durante a plataforma, maior será o seu tempo de transmissão.

O próximo indicador, utiliza o conceito de filas de mensagens e *multithreading*. O uso de filas de mensagens é vantajoso caso a taxa de aquisição do *software* de integração seja menor que a taxa de envio do sistema externo fonte. Consequentemente, para mitigar o caso de sobrecarga ao nível de mensagens acumuladas na fila é aplicado o conceito de *multithreading*. Desta forma, as mensagens da fila são processadas paralelamente, atenuando este *bottleneck*.

A última métrica é referente ao particionamento e balanceamento da carga. Este caso benéfico quando existem múltiplos canais em execução paralelamente. Quanto tal acontece, a máquina onde a plataforma executa fica exposta a um elevado esforço computacional, principalmente ao nível de uso de CPU, que se reflete na sua performance. De forma a reduzir esta utilização de recursos de *hardware*, é necessária a instanciação da plataforma, de modo a balancear a carga.

Questão 2: Qual a arquitetura ideal para a implementação de um sistema de integração baseado num motor de mensagens HL7?

Durante o desenvolvimento deste trabalho de investigação, a aplicação dos indicadores foi realizada consoante os cenários que representavam uma quebra no fluxo de mensagens.

Por exemplo, a aplicação de todos os indicadores apresentados para um cenário de teste onde o sistema externo fonte envia mensagens por uma conexão *TCP*, não apresenta necessariamente uma melhoria. Isto porque, caso a fonte envie continuamente um elevado volume de mensagens e no motor de integração as filas estiverem habilitadas, a taxa de acumulação de mensagens na fila será maior do que a taxa de processamento de mensagens do *software*, sendo mais benéfico desabilitar as filas (como pode ser observado pela figura 6.14). Desta forma, iria sobrecarregar em demasia a plataforma de integração, causando uma quebra na sua performance.

Em suma, a utilização de todas as métricas em simultâneo, apresentadas nesta dissertação, não significam forçosamente a melhor estrutura lógica. Inicialmente é necessário entender o volume de dados e a frequência que esse fluxo de dados ocorre em determinado ambiente, para posteriormente proceder à aplicação conveniente dos indicadores.

Questão 3: As ferramentas de avaliação usadas são escaláveis e aplicáveis a outros motores de integração?

Como já referenciado ao longo do documento, o *software* de integração utilizado foi o *Mirth Connect*. Esta é uma plataforma *open-source* que se apresenta como uma ferramenta robusta e abrangente no que diz respeito às funcionalidades e arquitetura da maioria dos sistemas de integração.

Normalmente, este tipo de *softwares* de integração, também designados de interface para assistência médica, apresentam uma arquitetura que suporta o fluxo de mensagens entre sistemas. Estas arquiteturas designam-se Enterprise Service Bus (ESB), que tem incorporados mecanismos de comunicação baseados numa infraestrutura Message Oriented Middleware (MOM), que permite a receção e o envio de mensagens. Por conseguinte, tanto filas de mensagens como mecanismos de *multithreading* representam uma caraterística que se encontram facilmente em sistemas que utilizam este tipo de arquiteturas.

Neste âmbito, o único indicador que poderá representar alguma complexidade quanto à sua implementação será o de instanciação através de uma ferramenta de virtualização, idêntico ao processo realizado neste trabalho. No entanto, este foi um método utilizado pelo motivo do ambiente de teste ser executado numa única máquina local, onde para uma execução em ambiente real, uma alternativa credível seria o particionamento da plataforma de integração por mais do que um servidor.

Para auxílio da realização de toda a dissertação, foram tidas em conta algumas publicações do fórum *Mirth Corporation* efetuadas por utilizadores de organizações que utilizam o *Mirth* e que são acompanhadas por desenvolvedores da plataforma [81]. Foi também publicado o método usado para aquisição de dados através de bases de dados *MongoDB*, possibilitando a sua partilha. Esta investigação apresenta um contributo para os utilizadores deste tipo de sistemas de software deterem noções de mecanismos que podem ser aplicados a fluxos de mensagens, em determinada unidade de saúde.

Perante todo o trabalho efetuado, pode-se concluir que os objetivos delineados para esta dissertação foram cumpridos através da extração bem sucedida de métricas que propiciam o bom desempenho de plataformas de integração. Relativamente à migração do ambiente de teste para um ambiente real não foi possível devido à situação pandémica atual, que se encontra instalada nas unidades de saúde.

Por toda a análise exercida ao longo desta investigação, constata-se que não existe uma estrutura lógica pré-definida que se possa ter em conta o bom desempenho de uma plataforma de integração. Neste contexto, inicialmente é necessário fazer o estudo do volume de mensagens implícito em determinado ambiente, e posteriormente, conforme a sua avaliação, determinar os melhores indicadores a aplicar.

7.1 TRABALHO FUTURO

Considerando o resultado final deste projeto de investigação, foi necessário fazer uma reflexão acerca dos aspetos que poderiam ser refinados e, possivelmente, o desenvolvimento de outras soluções que pudessem auxiliar tanto o processo de monitorização, como de melhoria de desempenho. Desta forma, como pontos que parecem relevantes para desenvolvimento posterior, propõe-se como trabalho futuro o seguinte:

- **Análise de outros tipos de comunicação:** neste trabalho foram contemplados conetores do tipo *readers* (bases de dados e *local file*) e *listeners* (comunicação via *TCP*). No entanto, ainda existem outros tipos comunicações incorporados nestes conetores, como comunicações via *HTTP*, por exemplo.

- **Desenvolvimento de uma interface que auxilie a monitorização da performance da plataforma de integração:** a criação de uma interface que possibilitasse a visualização de estatísticas acerca do desempenho de cada canal em execução. Esta ferramenta poderia apresentar informação como:
  - *Dashboards* com o número de mensagens enviadas, filtradas, em fila, ou recebidas;
  - Utilização de recursos utilizados, como CPU, pela plataforma de integração;
  - Recomendação de indicadores a utilizar, de acordo com o tempo que as mensagens recebidas demoram a ser processadas/enviadas;
  - Histórico do número médio de mensagens processadas por todos os canais, existindo um grau de comparação;

BIBLIOGRAFIA

[1] W. Wang, A. Tolk, and W. Wang, "The levels of conceptual interoperability model: Applying systems engineering principles to ms," in *Proceedings of the 2009 Spring Simulation Multiconference*, SpringSim '09, (San Diego, CA, USA), Society for Computer Simulation International, 2009.

[2] N. Theera-Ampornpunt, "Hl7 standards." https://www.slideshare.net/nawanan/hl7-standards-september-15-2016, 09 2016. (Accessed: 2019).

[3] "Hl7 message examples: version 2 and version 3." http://www.ringholm.com/docs/04300_en.htm. (Accessed: 2019).

[4] "Java messaging — jeyanthi thangiah." https://jthangiah.wordpress.com/2014/03/24/java-messaging/. (Accessed: 2020).

[5] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *J. Manage. Inf. Syst.*, vol. 24, p. 45–77, Dec. 2007.

[6] S. Agrawal, "Esb (enterprise service bus) architecture and implementation — hcl blogs." https://www.hcltech.com/blogs/everything-you-need-know-about-enterprise-service-bus-esb. (Accessed: 2020).

[7] J. Islam, "Container-based microservice architecture for local iot services." http://jultika.oulu.fi/files/nbnfioulu-201906072492.pdf, 2019. (Accessed: 2020).

[8] N. Healthcare, "Nextgen connect 3.7 user guide.pdf," 12 2018.

[9] "Iguana integration engine: Rapid, reliable integration. every time. - interfaceware inc." https://www.interfaceware.com/iguana.html. (Accessed: 2019).

[10] J. Weber-Jahnke, L. Peyton, and T. Topaloglou, "EHealth system interoperability," *Information Systems Frontiers*, vol. 14, no. 1, p. 3, 2012.

[11] R. Haux, "Health information systems - past, present, future," *International journal of medical informatics*, vol. 75, pp. 268–81, 03 2006.

[12] R. B. Cavalcante, M. N. Ferreira, and P. C. Silva, "Sistemas de Informação em Saúde: possibilidades e desafios," *Revista de Enfermagem da UFSM*, vol. 1, no. 2, p. 290, 2011.

[13] F. Marins, L. Cardoso, F. Portela, M. F. Santos, A. Abelha, and J. Machado, "Improving high availability and reliability of health interoperability systems," *Advances in Intelligent Systems and Computing*, vol. 276, pp. 207–216, 2014.

[14] H. Peixoto, M. Santos, A. Abelha, and J. Machado, "Intelligence in interoperability with AIDA," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7661 LNAI, pp. 264–273, 2012.

[15] M. Soto, C. Sicotte, and A. Motulsky, "Using Health Information Exchange: Usability and Usefulness Evaluation," *Studies in health technology and informatics*, vol. 264, pp. 1036–1040, 2019.

[16] S. Bhartiya and D. Mehrotra, "Exploring interoperability approaches and challenges in healthcare data exchange," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8040 LNCS, pp. 52–65, 2013.

[17] H. Pirnejad, R. Bal, and M. Berg, "Building an inter-organizational communication network and challenges for preserving interoperability," *International Journal of Medical Informatics*, vol. 77, no. 12, pp. 818 – 827, 2008.

[18] H. Peixoto, J. Machado, J. Neves, and A. Abelha, "Semantic Interoperability and Health Records," in *E-Health* (H. Takeda, ed.), (Berlin, Heidelberg), pp. 236–237, Springer Berlin Heidelberg, 2010.

[19] G. J. Joyia, M. U. Ak, C. N. Akbar, and M. F. Maqsood, "Evolution of Health Level-7: A survey," *ACM International Conference Proceeding Series*, no. July, pp. 118–123, 2018.

[20] J. Lin, F. Shi, M. Figurski, K. Ranslam, and Z. Liu, "Data migration from operating EMRs to OpenEMR with mirth connect," *Studies in Health Technology and Informatics*, vol. 257, pp. 288–292, 2019.

[21] NextGen Healthcare, "Interface Engines Simplify Interoperability - But Should You Go It Alone?," 01 2018.

[22] "Who we are, a healthcare solutions provider." https://www.nextgen.com/about-us. (Accessed: 2019).

[23] A. Castanheira, H. Peixoto, and J. Machado, "Overcoming challenges in healthcare interoperability regulatory compliance," in *Ambient Intelligence – Software and Applications* (P. Novais, G. Vercelli, J. L. Larriba-Pey, F. Herrera, and P. Chamoso, eds.), (Cham), pp. 44–53, Springer International Publishing, 2021.

[24] C. Brito, M. Esteves, H. Peixoto, A. Abelha, and J. Machado, "A data mining approach to classify serum creatinine values in patients undergoing continuous ambulatory peritoneal dialysis," *Wireless Networks*, 01 2019.

[25] C. Peixoto, C. Brito, M. Fontainhas, A. Abelha, H. Peixoto, and J. Machado, "Continuous ambulatory peritoneal dialysis: Business intelligence applied to patient monitoring: Capd study and statistics," in *2017 5th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pp. 178–185, IEEE, 2017.

[26] M. Hammadeh, "The current state of interoperability — himss." https://www.himss.org/news/current-state-interoperability, 06 2018. (Accessed: 2019).

[27] "Solving the interoperability challenge - ieee pulse." https://pulse.embs.org/november-2014/solving-interoperability-challenge/. (Accessed: 2019).

[28] "Interoperability, federal communications commission." https://www.fcc.gov/general/interoperability. (Accessed: 2019).

[29] "Cross-domain interoperability – ncoic." https://www.ncoic.org/cross-domain-interoperability/. (Accessed: 2019).

[30] A. Tolk, S. Diallo, and C. Turnitsa, "Applying the levels of conceptual interoperability model in support of integratability, interoperability, and composability for system-of-systems engineering," *International Journal Systemics, Cybernetics and Informatics*, vol. 5, 10 2007.

[31] E. H. Page, R. Briggs, and J. A. Tufarolo, "Toward a family of maturity models for the simulation interconnection problem," in *Proceedings of the Spring Simulation Interoperability Workshop*, vol. 1, pp. 1059–1069, IEEE Computer Society, 2004.

[32] "3 barriers to interoperability, and 3 reasons it matters." https://bridgeconnector.co/resources/#blogs. (Accessed: 2019).

[33] A. J. Holmgren, V. Patel, and J. Adler-Milstein, "Progress in interoperability: Measuring us hospitals' engagement in sharing patient data," *Health Affairs*, vol. 36, no. 10, pp. 1820–1827, 2017. PMID: 28971929.

[34] "Council post: The state of interoperability in healthcare." https://www.forbes.com/sites/forbestechcouncil/2018/05/31/the-state-of-interoperability-in-healthcare/, 05 2018. (Accessed: 2019).

[35] "4 reasons why ehr interoperability is a mess (and how to fix it) — datica blog." https://datica.com/blog/reasons-ehr-interoperability-is-a-mess-and-how-to-fix-it/, 06 2019. (Accessed: 2019).

[36] "Top 5 challenges to achieving healthcare interoperability." https://ehrintelligence.com/news/top-5-challenges-to-achieving-healthcare-interoperability#. (Accessed: 2019).

[37] "About health level seven international — hl7 international." http://www.hl7.org/about/index.cfm?ref=nav. (Accessed: 2019).

[38] "O hl7." http://interopera.esy.es/wp-content/uploads/2017/04/O-HL7-PRIME-1.pdf. (Accessed: 2019).

[39] "Hl7 standards - section 1: Primary standards — hl7 international." http://www.hl7.org/implement/standards/product_section.cfm?section=1. (Accessed: 2019).

[40] "Hl7 standards product brief - hl7 version 3 product suite — hl7 international." http://www.hl7.org/implement/standards/product_brief.cfm?product_id=186. (Accessed: 2019).

[41] R. Spronk, "Hl7 version 3: Message or cda document?." http://www.ringholm.de/docs/04200_en.htm, 09 2007. (Accessed: 2019).

[42] "Index - fhir v4.0.1." https://www.hl7.org/fhir/, October 2019. (Accessed: 2019).

[43] "Trigger event - hl7wiki." https://wiki.hl7.org/index.php?title=Trigger_Event. (Accessed: 2019).

[44] R. Guziołowski, "Design and implementation of fully configurable interpreter and generator of hl7 standard protocol messages," *Poznan University of Technology technical report no. RA-xxx/06*, 08 2006.

[45] "Integration engines all software product ranking comparison." https://klasresearch.com/compare/integration-engines/19. (Accessed: 2019).

[46] R. Alemy, "Mirth connect introduction and tutorial [patesco]." http://wiki.patesco.ca/doku.php?id=hl7:mirth:tutorial#mirth_connect_introduction_and_tutorial, 11 2018. (Accessed: 2019).

[47] "Why choose mirth connect as your hl7 interface engine? — technosoft solutions." https://techno-soft.com/why-choose-mirth-connect-as-your-hl7-interface-engine.html/, 12 2018. (Accessed: 2019).

[48] "Corepoint and rhapsody merge to advance interoperability in healthcare - corepoint health." https://corepointhealth.com/rhapsody-and-corepoint-merge-to-advance-interoperability-in-healthcare/, 07 2019. (Accessed: 2019).

[49] C. Health, "Corepoint Integration Engine." https://docplayer.net/43606291-Corepoint-integration-engine.html, 2009. (Accessed: 2019).

[50] "Corepoint integration engine - product overview - corepoint health." https://corepointhealth.com/resource-center/datasheets/corepoint-integration-engine-product-overview/. (Accessed: 2019).

[51] i. Inc, "What is a channel?." http://www.interfaceware.com/manual/what_is_a_channel_.html. (Accessed: 2019).

[52] J. Le Boudec, "Performance evaluation of computer and communication systems." https://researchswinger.org/others/perf.pdf, 05 2007.

[53] E. Waldman, "Usos da vigilância e da monitorização em saúde pública," *Informe Epidemiológico do SUS*, vol. 7, 09 1998.

[54] "Cpu analysis — microsoft docs." https://docs.microsoft.com/en-us/windows-hardware/test/wpt/cpu-analysis, 05 2017. (Accessed: 2019).

[55] "Results for the memory footprint assessment — microsoft docs." https://docs.microsoft.com/en-us/windows-hardware/test/assessments/results-for-the-memory-footprint-assessment, 05 2017. (Accessed: 2019).

[56] "Getdiskfreespaceexa function (fileapi.h) - win32 apps — microsoft docs." https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-getdiskfreespaceexa, 05 2018. (Accessed: 2019).

[57] "Introduction to message queuing." https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.pro.doc/q002620_.htm. (Accessed: 2020).

[58] P. Pietzuch, D. Eyers, S. Kounev, and B. Shand, "Towards a common api for publish/-subscribe," in *Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems*, (New York, NY, USA), p. 152–157, Association for Computing Machinery, 2007.

[59] J. S. Ward and A. Barker, "Undefined by data: A survey of big data definitions," 2013.

[60] M. Stonebraker, U. Cetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *SIGMOD Record*, vol. 34, pp. 42–47, 12 2005.

[61] S. Chen and P. Greenfield, "Qos evaluation of jms: an empirical approach," *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*, pp. 10 pp.–, 2004.

[62] P. Maheshwari and M. Pang, "Benchmarking message-oriented middleware: Tib/rv versus sonicmq," *Concurrency - Practice and Experience*, vol. 17, pp. 1507–1526, 10 2005.

[63] P. Dobbelaere and K. S. Esmaili, "Kafka versus rabbitmq," *ArXiv*, vol. abs/1709.00333, 2017.

[64] S. March and V. Storey, "Design science in the information systems discipline: An introduction to the special issue on design science research," *MIS Quarterly*, vol. 32, 12 2008.

[65] A. Elragal and M. Haddara, "Design science research: Evaluation in the lens of big data analytics," *Systems*, vol. 7, p. 27, May 2019.

[66] V. Vaishnavi, B. Kuechler, and S. Petter, "Design science research in information systems." http://www.desrist.org/design-research-in-information-systems. (Accessed: 2020).

[67] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *Management Information Systems Quarterly*, vol. 28, pp. 75–, 03 2004.

[68] H. S. Oluwatosin, "Client-server model," *IOSR J Comput Eng (IOSR-JCE)*, vol. 16, no. 1, p. 67, 2014.

[69] G. Bortis, "Experiences with mirth: An open source health care integration engine," in *Proceedings of the 30th International Conference on Software Engineering*, p. 649–652, 2008.

[70] F. Menge, "Enterprise service bus," in *Free and open source software conference*, vol. 2, pp. 1–6, 2007.

[71] "Mirth connect 3.7.0 - what's new - mirth connect - confluence." http://www.mirthcorp.com/community/wiki/display/mirth/Mirth+Connect+3.7.0+-+What%27s+New. (Accessed: 2020).

[72] N. Jatana, S. Puri, M. Ahuja, I. Kathuria, and D. Gosain, "A survey and comparison of relational and non-relational database," *International Journal of Engineering Research & Technology*, vol. 1, no. 6, pp. 1–5, 2012.

[73] "Overview of pl/sql." https://docs.oracle.com/cd/E11882_01/appdev.112/e25519/overview.htm#LNPLS00103. (Accessed: 2020).

[74] C. Győrödi, R. Győrödi, G. Pecherle, and A. Olah, "A comparative study: Mongodb vs. mysql," in *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, pp. 1–6, 2015.

[75] "Advantages of mongodb — mongodb." https://www.mongodb.com/advantages-of-mongodb. (Accessed: 2020).

[76] "Docker documentation — docker documentation." https://docs.docker.com/. (Accessed: 2020).

[77] "Apache jmeter - apache jmeter™." https://jmeter.apache.org/. (Accessed: 2019).

[78] D. Nevedrov, "Using jmeter to performance test web services," *Published on dev2dev*, 2006.

[79] C. Truica, F. Radulescu, A. Boicea, and I. Bucur, "Performance evaluation for crud operations in asynchronously replicated document oriented database," in *2015 20th International Conference on Control Systems and Computer Science*, pp. 191–196, 2015.

[80] V. Kasavajhala, "Solid state drive vs. hard disk drive price and performance study," *Proc. Dell Tech. White Paper*, pp. 8–9, 2011.

[81] "Mirth connect - mirth community." https://forums.mirthproject.io/forum/mirth-connect?f=3. (Accessed: 2019).

A

ANEXOS AUXILIARES DA ARQUITETURA DO SISTEMA

```
//Conexao a base de dados MySQL
var con = mysql.createConnection({
host: "ipadress",
port: port,
user: "root",
password: "password",
database:"dbteste"
});

//Metodo que permite a execucao do 'Insert' na base de dados de X em X
    milissegundos
function sleep(ms) {
  return new Promise(
    resolve => setTimeout(resolve, ms)
  );
}

//Metodo para inserir dados aleatorios na bases de dados
con.connect(async function(err) {
  if (err) throw err;
  console.log('Connect!')
  var counter = 0
  for(var j=0; j<62; j++){
    await sleep(30000)
    //Query insert executada 400 vezes de 30-30 segundos
    for(var i = 0; i<400; i++)-{
      counter++
      var nomefich = random.int(min = 1,100000000)
      var episode = random.int(min = 10000000, max = 999999999)
      con.query("INSERT INTO hl7envia(origem, destino, msgid, nomefich,
                evn, episodio, status) VALUES ('source', 'destiny',
                " + counter + ", 'FileNumber_"+ nomefich +"', 'ADT_019',
                "+ episode +", 0)",
      function (err, result, fields){
        if (err) throw err;
      })
    }
  }
});
```

Excerto de código A.1: *Script NodeJS* para geração de dados.

```
//Criacao da lista a adicao do conteudo das variaveis mapeadas
var params = new Packages.java.util.ArrayList();
params.add($('origem'));
params.add($('destino'));
params.add($('msgid'));
params.add($('datager'));

var sql = "INSERT INTO hl7analisa (Origem, Destino, Msgid, DataGer,
           DataEnv, ErroHl7) VALUES (?, ?, ?, sysdate(), null)";

//Conexao a base de dados MySQL e insercao dos dados na tabela conforme a
    query e a lista
try {
  var dbConn = DatabaseConnectionFactory.createDatabaseConnection(
  'com.mysql.cj.jdbc.Driver','jdbc:mysql://ipadress:port/dbteste',
  'root','password');

  dbConn.executeUpdate(sql, params);
} finally {
  if (dbConn) {
    dbConn.close();
  }
}
```

Excerto de código A.2: Etapa *JavaScript* do *Source Transformer*.

```
if (responseStatus == ERROR) {
  var params = new Packages.java.util.ArrayList();
  params.add($('MsgId'));

  var sql = "UPDATE hl7envia set ERROHL7 = 1 where MSGID = ?";

  try {
    var dbConn = DatabaseConnectionFactory.createDatabaseConnection(
    'com.mysql.cj.jdbc.Driver','jdbc:mysql://ipadress:port/dbteste',
    'root','password');

    dbConn.executeUpdate(sql, params);
    } finally {
      if (dbConn) {
        dbConn.close();
      }
    }
}

if (responseStatus == SENT) {
  router.routeMessageByChannelId('527d9686-5d7d-40d0-b300-7faf5d8d9bb4',
                                                    response.getMessage());
}
```

Excerto de código A.3: Etapa *JavaScript* do *Destination Response Transformer*.

```
INSERT INTO hl7recebe(Origem,Destino,Evn,DataGer,Episodio)
VALUES
(${origem},${destino},${Evn},${DataGer},${Episodio})
```

Excerto de código A.4: Query *SQL* para inserção de dados.

```
try {
  //Estabele a conexao
  var mongoClient = new Packages.com.mongodb.MongoClient("ipadress", port);
  var database = mongoClient.getDatabase("dbteste");
  var collection = database.getCollection("hl7recebe");

  //Atribuicao dos valores das variaveis mapeadas, aos campos de um
      documento
  var jsonDoc = JSON.stringify({ "Origem" : $("origem"),
                                 "Destino" : $("destino"),
                                 "Evn" : $("evn"),
                                 "DataGer" : $("datager"),
                                 "Episodio" : $("episodio")
                               });
  //Insercao do documento na BD
  var doc = Packages.org.bson.Document.parse(jsonDoc);
  collection.insertOne(doc);

  return;
} finally {
  if (mongoClient) {
    mongoClient.close();
  }
}
```

Excerto de código A.5: Conexão ao MongoDB e inserção de documentos.

B

RESULTADOS DOS TESTES DE FLUXO DE MENSAGENS

	Resultados iniciais com diferentes fontes/destinos	
	Sistema Fonte	**Sistema Destino**
MySQL	22214	22214
Oracle	22633	22092
MongoDB	23884	22444
Local File	23930	23382
TCP	24466	23563

Tabela B.1: Número de mensagens com diferentes sistemas fonte/destino.

	Introdução de um componente de filtragem, para SQL *Source Databases*			
	Com Filtro			Sem Filtro
	Received	**Filtered**	**Sent**	**Received/Sent**
MySQL	12532	5360	7172	7929
Oracle	11059	3400	7659	8248

Tabela B.2: Número de mensagens com/sem componentes de filtragem.

	Diferentes modos de armazenamento				
	Development	**Production**	**Raw**	**Metadata**	**Disable**
MySQL	7920	8066	8636	8800	11812
Oracle	8248	8347	8761	8935	12006
MongoDB	8622	8828	9592	9783	12894
TCP	8945	9175	9835	10077	13199

Tabela B.3: Número de mensagens com diferentes modos de armazenamento.

	1 instância *Mirth* com 1 *Receiver Channel*	
	1 Thread	**4 Threads**
MySQL	7920	7964
Oracle	8248	8281
MongoDB	8622	8642
TCP	8945	14642

Tabela B.4: Número de mensagens sem/com *multithreading*.

	1 instância *Mirth* com 1 *Receiver Channel*					
	1 Thread			**4 Threads**		
	Received	**Queued**	**Sent**	**Received**	**Queued**	**Sent**
MySQL	24247	14511	9736	24073	9495	14578
Oracle	24104	14270	9834	24000	8217	15783
MongoDB	24291	15590	8701	24356	8556	15800
TCP	80587	73365	7310	123688	107737	15951

Tabela B.5: Número de mensagens com filas ativas e sem/com *multithreading*.

		1 instância *Mirth* com 2 *Receiver Channels*	
		1 Thread	**4 Threads**
MySQL	***Receiver 1***	6200	6412
	Receiver 2	6208	6430
	Total	12408	12842
Oracle	***Receiver 1***	6631	6714
	Receiver 2	6657	6738
	Total	13288	13452
MongoDB	***Receiver 1***	6978	7070
	Receiver 2	6959	7093
	Total	13937	14163
TCP	***Receiver 1***	6504	7291
	Receiver 2	6504	7292
	Total	13008	14583

Tabela B.6: Número de mensagens sem/com *multithreading* (2 *receivers*).

		1 instância *Mirth* com 2 *Receiver Channels*					
		1 Thread			4 Threads		
		Received	Queued	Sent	Received	Queued	Sent
MySQL	*Receiver 1*	24292	18237	6055	24244	18126	6118
	Receiver 2	24361	18289	6072	24219	18053	6166
	Total	48653	36526	12127	48463	36179	12284
Oracle	*Receiver 1*	23272	16804	6468	23717	17193	6524
	Receiver 2	23493	16993	6500	23750	17217	6533
	Total	46765	34089	12968	47467	33010	13057
MongoDB	*Receiver 1*	24264	17949	6315	24312	17708	6604
	Receiver 2	24241	17926	6315	24187	17389	6798
	Total	48505	35875	12630	48499	35097	13402
TCP	*Receiver 1*	60784	54940	5844	98192	91142	7050
	Receiver 2	60809	54966	5843	98413	91358	7055
	Total	121593	109906	11687	196605	182500	14105

Tabela B.7: Número de mensagens com filas ativas e sem/com *multithreading* (2 *receivers*).

		3 instâncias *Mirth* com 2 *Receiver Channels* e 1 *Channel Destination*	
		1 Thread	4 Threads
MySQL	Instância 1	6991	6921
	Instância 2	7050	6916
	Instância 3	14041	13837
Oracle	Instância 1	7066	6988
	Instância 2	7097	7009
	Instância 3	14163	13997
MongoDB	Instância 1	7510	7575
	Instância 2	7502	7604
	Instância 3	15012	15179
TCP	Instância 1	6808	11358
	Instância 2	6791	11340
	Instância 3	13599	22698

Tabela B.8: Número de mensagens sem/com *multithreading* (3 instâncias *Mirth*).

		3 instâncias Mirth com 2 *Receiver Channels* e 1 *Channel Destination*					
		1 Thread			4 Threads		
		Received	**Queued**	**Sent**	**Received**	**Queued**	**Sent**
MySQL	**Instância 1**	24171	18495	5676	24057	14205	9852
	Instância 2	24227	18424	5803	24188	14319	9869
	Instância 3	11379	—	11379	19721	—	19721
Oracle	**Instância 1**	21894	15329	6565	23139	13339	9800
	Instância 2	22103	15482	6621	23419	13537	9882
	Instância 3	13186	—	13186	19682	—	19682
MongoDB	**Instância 1**	24400	18500	5900	24536	14144	10392
	Instância 2	24402	18460	5942	24348	13905	10443
	Instância 3	11842	—	11842	20736	—	20736
TCP	**Instância 1**	45818	40903	7915	70664	61152	9512
	Instância 2	45786	40848	4938	70508	61002	9506
	Instância 3	9853	—	9853	19018	—	19018

Tabela B.9: Número de mensagens com filas ativas e sem/com *multithreading* (3 instâncias Mirth).

C

RESULTADOS DO DOCKER STATS

		Utilização de Recursos (1 Instância *Mirth*, 1 *Receiver*)			
		CPU (%)	**Memory (MB/s)**	**Block IO (MB)**	**Network IO (MB)**
MySQL	**1 Thread**	42.09	557	34.3/0.0164	187/537
	4 Threads	44.40	541.7	27.3/0.0164	183/518
Oracle	**1 Thread**	42.94	513.9	47.2/0.0164	189/530
	4 Threads	44.58	464.2	32.8/0.0164	187/538
MongoDB	**1 Thread**	44.24	541.4	40.9/0.0164	203/578
	4 Threads	45.82	606	11.9/0.0164	219/572
TCP	**1 Thread**	38.10	887.2	44/0.0164	374/776
	4 Threads	53.54	913	20.7/0.0164	406/978

Tabela C.1: Resultados *Docker Stats* - 1 Instância sem/com *multithreading*.

		Utilização de Recursos (1 Instância *Mirth*, 1 *Receiver*, filas ativas)			
		CPU (%)	**Memory (MB/s)**	**Block IO (MB)**	**Network IO (MB)**
MySQL	**1 Thread**	45.57	727	15.7/0.0164	362/748
	4 Threads	49.65	642.9	20.4/0.0164	443/1030
Oracle	**1 Thread**	47.74	944	51/0.0164	397/853
	4 Threads	49.68	738	61.7/0.0164	483/1130
MongoDB	**1 Thread**	47.91	1154	11.9/0.0164	295/715
	4 Threads	54.31	721.8	34.7/0.0164	448/1130
TCP	**1 Thread**	55.65	1150	46/0.0164	566/956
	4 Threads	55.23	1049	53.1/0.0164	970/1590

Tabela C.2: Resultados *Docker Stats* - 1 Instância com filas ativas sem/com *multithreading*.

		Utilização de Recursos (1 Instância *Mirth*, 2 *Receivers*)			
		CPU (%)	**Memory (MB/s)**	**Block IO (MB)**	**Network IO (MB)**
MySQL	**1 Thread**	58.76	604.6	33.4/0.0164	300/851
	4 Threads	60.81	554.5	33.3/0.0164	298/838
Oracle	**1 Thread**	53,37	771	13.8/0.0164	309/889
	4 Threads	72.46	744	22.6/0.0164	309/866
MongoDB	**1 Thread**	56.48	906	41.4/0.0164	333/931
	4 Threads	61.34	863.1	26.07/0.0164	340/947
TCP	**1 Thread**	45.53	694.5	30.6/0.0164	375/968
	4 Threads	43.56	901.5	34.2/0.0164	407/1078

Tabela C.3: Resultados *Docker Stats* - 1 Instância sem/com *multithreading* (2 *receivers*).

		Utilização de Recursos (1 Instância *Mirth*, 2 *Receivers*, filas ativas)			
		CPU (%)	**Memory (MB/s)**	**Block IO (MB)**	**Network IO (MB)**
MySQL	**1 Thread**	69.62	787.7	16.4/0.0164	454/980
	4 Threads	75.02	644.2	12.3/0.0164	584/1070
Oracle	**1 Thread**	70.19	1236	27.22/0.0164	551/1160
	4 Threads	76.47	1194	22.06/0.0164	550/1190
MongoDB	**1 Thread**	74.24	1423	69/0.0164	535/1080
	4 Threads	76.24	1235	57.58/0.0164	473/1110
TCP	**1 Thread**	66.34	922.9	18.1/0.0164	875/1320
	4 Threads	74.45	1108	53.7/0.0164	1190/1610

Tabela C.4: Resultados *Docker Stats* - 1 Instância com filas ativas e sem/com *multithreading* (2 *receivers*).

		Utilização de Recursos (3 Instâncias *Mirth*)			
		1 Thread			
		CPU (%)	Memory (MB/s)	Block IO (MB)	Network IO (MB)
MySQL	Instância 1	18.70	819	19.6/0.0164	119/356
	Instância 2	18.64	813.5	23.85/0.0164	115/340
	Instância 3	27.53	721.6	18.5/0.0164	108/235
		4 Threads			
	Instância 1	18.91	858	33/0.0164	118/348
	Instância 2	19.04	869	30.6/0.0164	116/346
	Instância 3	29.34	710.7	20.8/0.0164	108/235
Oracle		1 Thread			
	Instância 1	19.40	992.3	43.6/0.0164	120/358
	Instância 2	18.84	950	39.8/0.0164	119/354
	Instância 3	28.87	1155	59/0.0164	111/241
		4 Threads			
	Instância 1	22.16	873.9	35.8/0.0164	117/342
	Instância 2	20.73	837	40.4/0.0164	117/349
	Instância 3	31.17	1465	117/0.0164	110/238
MongoDB		1 Thread			
	Instância 1	29.70	1163	37.2/0.0164	130/367
	Instância 2	20.64	1120	48.5/0.0164	128/373
	Instância 3	8.85	988.6	52/0.0164	117/253
		4 Threads			
	Instância 1	19.84	1210	32.5/0.0164	132/375
	Instância 2	20.72	982.5	44.7/0.0164	129/372
	Instância 3	31.04	738.2	39.7/0.0164	119/157
TCP		1 Thread			
	Instância 1	17.64	859	39.3/0.0164	161/393
	Instância 2	17.17	810	35.9/0.0164	161/390
	Instância 3	8.40	720.9	31/0.0164	64.5/207
		4 Threads			
	Instância 1	26.63	529.9	53/0.0164	201/600
	Instância 2	27.47	664.3	51/0.0164	200/598
	Instância 3	11.15	645.9	26.33/0.0164	109/332

Tabela C.5: Resultados *Docker Stats* - 3 Instâncias sem/com *multithreading*.

		Utilização de Recursos (3 Instâncias *Mirth*, filas ativas)			
		1 Thread			
		CPU (%)	Memory (MB/s)	Block IO (MB)	Network IO (MB)
MySQL	Instância 1	23.49	971	31/0.0164	155/363
	Instância 2	24.04	925	35.8/0.0164	141/361
	Instância 3	24.63	794	24.3/0.0164	90.3/197
		4 Threads			
	Instância 1	31.88	1078	40.8/0.0164	204/548
	Instância 2	32.01	1044	45.8/0.0164	203/542
	Instância 3	33.86	1014	32.3/0.0164	154/327
Oracle		1 Thread			
	Instância 1	23.39	1059.1	47.5/0.0164	191/419
	Instância 2	23.51	1026	41.8/0.0164	192/422
	Instância 3	24.41	904.6	31/0.0164	104/226
		4 Threads			
	Instância 1	31.17	1046.6	40.6/0.0164	221/552
	Instância 2	31.64	1063.7	43.7/0.0164	221/557
	Instância 3	35.29	1251	31.4/0.0164	155/328
MongoDB		1 Thread			
	Instância 1	24.7	906.7	35/0.0164	156/370
	Instância 2	24.14	945.5	33.5/0.0164	158/381
	Instância 3	27.42	1170	25/0.0164	92.9/202
		4 Threads			
	Instância 1	28.63	1141	45/0.0164	228/528
	Instância 2	28.37	1220	42.4/0.0164	229/592
	Instância 3	36.36	1050	24.1/0.0164	163/346
TCP		1 Thread			
	Instância 1	29.16	1282	28/0.0164	331/499
	Instância 2	29.65	1150	33.2/0.0164	326/494
	Instância 3	6.64	609.2	26.12/0.0164	49/156
		4 Threads			
	Instância 1	40.15	947.4	53.9/0.0164	537/808
	Instância 2	42.83	904.5	59.1/0.0164	532/803
	Instância 3	12.86	705.7	56.66/0.0164	91.3/281

Tabela C.6: Resultados *Docker Stats* - 3 Instâncias com filas ativas e sem/com *multithreading*.