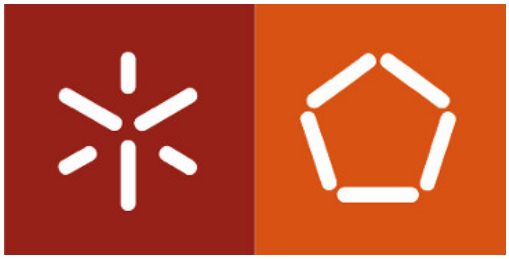

Universidade do Minho

Mestrado integrado em Engenharia Informática

F01 - Interoperabilidade com Hl7 e Mirth

Grupo 5

Vasco António Lopes Ramos, PG42852

Interoperabilidade Semântica

4º Ano, 2º Semestre

Departamento de Informática

29 de abril de 2021

Índice

Lista de Figuras

Lista de Códigos

1 Introdução

1.1 Contextualização

No âmbito da UC de *Interoperabilidade Semântica* foi-nos proposta a realização de uma primeira ficha de exercícios para explorar a aplicação de interoperabilidade semântica através de mensagens HL7 no contexto de um sistema de requisição e execução de exames médicos. De forma mais explícita, o objetivo deste trabalho era desenvolver um ambiente com 2 sistemas completamente separados que comunicam as suas ações entre si através de HL7, usando o Mirth Connector.

Este trabalho foi desenvolvido pelos dois elementos do grupo (Grupo 05), ao longo de 3 semanas, em que na primeira semana se começou por desenvolver as bases de dados e as funcionalidades das aplicações clientes de cada serviço e a geração das respetivas mensagens HL7; na segunda semana procedeu-se ao desenvolvimento dos canais recetores das mensagens através do *Mirth* e da inclusão da admissão de pacientes como uma funcionalidade extra. Por fim, na terceira semana procedeu-se à escrita do presente relatório.

Ao nível da divisão de trabalho: eu, Vasco Ramos, fiquei responsável pelo desenvolvimento de tudo o que estava relacionado com o serviço 1 e a minha colega, Carolina Marques, ficou responsável pelo serviço 2.

1.2 Estrutura do relatório

Este relatório encontra-se dividido em 5 partes:

No primeiro capítulo é feita uma breve introdução a este trabalho, e é explicada a estrutura deste relatório.

No segundo capítulo é feita uma análise sobre a arquitetura geral do sistema implementado, bem como as tecnologias usadas para as diferentes camadas.

No terceiro capítulo são abordados tópicos relevantes sobre decisões e estruturas de implementação relativas às três principais camadas: base de dados, clientes dos serviços e comunicação entre serviços.

No quarto capítulo procede-se à exposição e demonstração dos resultados obtidos, bem como uma discussão dos mesmos.

Por fim, no último capítulo é feito um breve resumo do que foi possível implementar e realizar ao longo do trabalho respetivo a este relatório, bem como uma breve análise conclusiva sobre o que se retirou do mesmo.

2 Arquitetura

Devido ao confinamento relacionado com a pandemia Covid, foi impossível estar fisicamente junto com a minha colega, pelo que teve de ser encontrada uma alternativa para a comunicação entre os dois serviços. Desta forma, para simular que os serviços estavam instalados em máquinas diferentes, criou-se duas bases de dados diferentes, cada uma com um utilizador que só tem permissões para aceder à sua base de dados e um servidor *Mirth* com dois *channels* diferentes, um para cada serviço.

Ao nível dos diferentes componentes da arquitetura, temos o servidor *Mirth*, os clientes de cada serviço, desenvolvidos em Python e as base de dados MySQL.

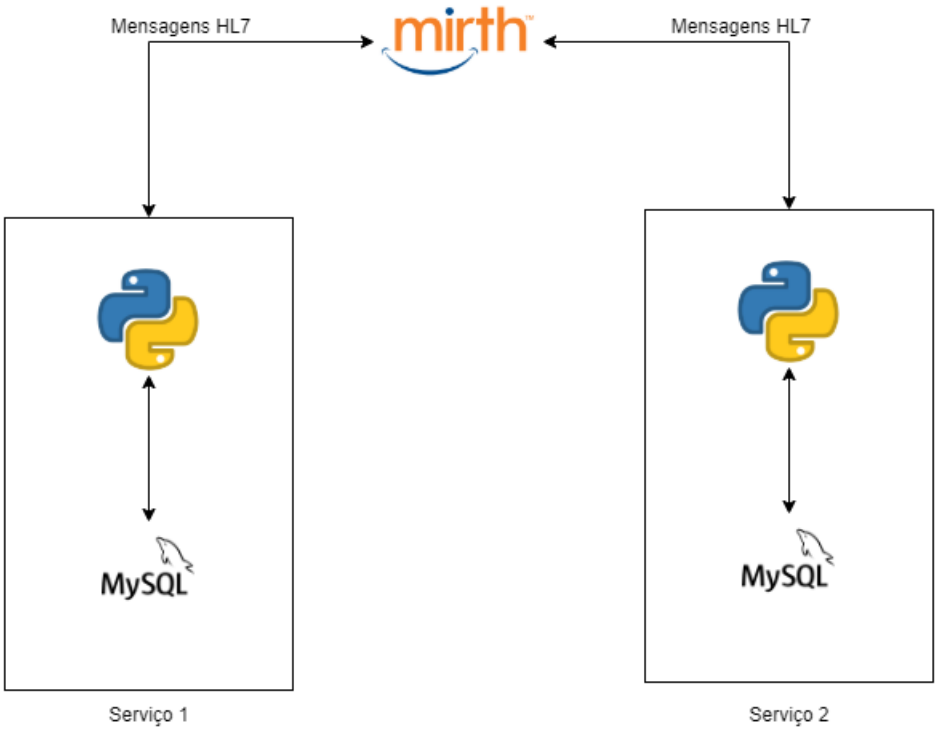

Figura 1: Visão simplificada da arquitetura

A figura 1 apresenta uma visão simplificada de como os vários componentes se relacionam. Mais detalhadamente, o utilizador interage, diretamente, com o serviço 1 através do seu menu interativo, desenvolvido em *Python*. Este, aquando

de novas ações que o justifiquem interage com a base de dados inserindo ou atualizando informação, reporta essas ações ao serviço 2 por mensagens HL7 via *Mirth* e guarda essas mensagens no sistema de ficheiros como *logs* e *backup*. No serviço 2 o processo de ação é semelhante. Quando o *channel* do *Mirth* respetivo ao serviço 1 recebe mensagens HL7 (relativas a ações no serviço 2), estas são analisadas e as ações são replicadas na base de dados do serviço 1. O mesmo acontece no serviço 2.

2.1 Escolha de Tecnologias

2.1.1 Bases de Dados

Para as bases de dados que suportam os serviços desenvolvidos decidiu-se usar **MySQL**, por ser simples e rápido de usar, bem como pelo contacto prévio e familiaridade com este motor de base de dados.

2.1.2 Clientes (serviços)

Os clientes (e respetivos serviços) foram desenvolvidos em **Python** com o apoio de duas bibliotecas: *HL7apy*, [1], para a construção e validação das mensagens HL7 e *python-hl7*, [2], para o envio das mensagens para o servidor Mirth (via TCP MLLP).

2.1.3 Comunicação entre serviços

Ao nível da comunicação entre os dois serviços usou-se **HL7** como protocolo de mensagens e o **Mirth** como servidor de comunicação entre os mesmos (via TCP MLLP). Para a implementação das mensagens HL7 baseámo-nos nos slides disponibilizados pelo docente da UC e documentação *online* [3].

3 Implementação

3.1 Bases de Dados

Ao nível das bases, como se vai ver a seguir, para além de usar duas bases de dados separadas para cada serviço, a nomenclatura das tabelas e respetivas colunas de cada base de dados difere, bem como a própria organização dos dados, que não é exatamente igual nas duas bases de dados.

3.1.1 Serviço 1

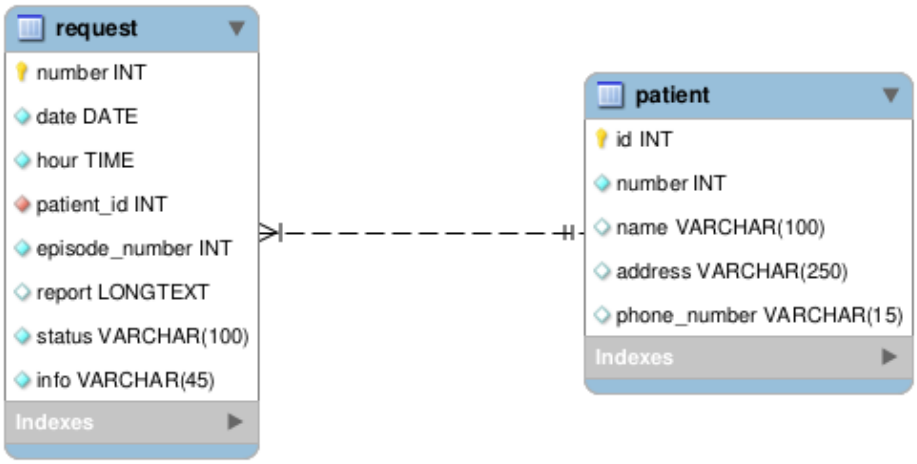

Figura 2: Serviço 1 - Esquema da Base de Dados

Como se pode ver na figura 2, o serviço 1 utiliza uma base de dados com o nome de ***medical_exams***. A informação dos pacientes é guardada na tabela ***patient***, sendo que os seus valores podem, ou não ser atualizados consoante a utilização do programa. A informação relativa aos pedidos de exames é guardada na tabela ***request***.

3.1.2 Serviço 2

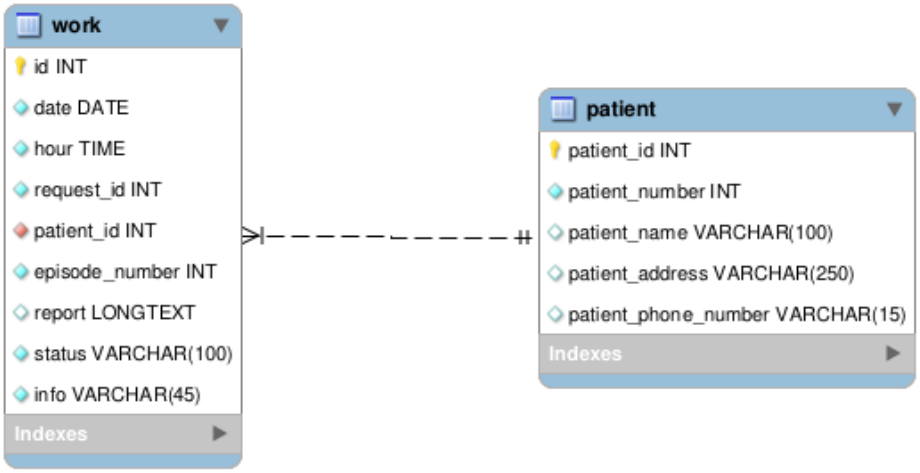

Figura 3: Serviço 2 - Esquema da Base de Dados

Tal como mostra a figura 3, o serviço 2 utiliza uma base de dados com o nome de ***medical_work_list***. A informação dos pacientes é guardada na tabela ***patient***, sendo que os seus valores podem, ou não ser atualizados consoante a utilização do programa. Por fim, a informação relativa aos pedidos de exames é guardada na tabela ***work***.

3.2 Clientes (serviços)

Tal como referido na secção 2.1, os clientes para os serviços foram desenvolvidos em **Python** com o apoio de duas bibliotecas: *HL7apy* e *python-hl7*. Devido à considerável dificuldade em utilizar a biblioteca *HL7apy*, pela sua complexidade e falta de documentação útil, decidiu-se criar uma camada de abstração adicional à utilização da mesma, para facilitar a construção dinâmica das mensagens **HL7**, pelo que foram criadas funções para gerar as mensagens necessárias de forma dinâmica, tendo em conta os valores de input, passados por argumentos.

O pedaço de código 1 mostra a função *high-level* que foi criada para abstrair a construção das mensagens que depois desagua em funções focalizadas para cada tipo de mensagem necessária aos serviços desenvolvidos.

```
def generate_hl7_message(type, sender, receiver, data, op=0):
  if type == "ORM_O01":
    return generate_hl7_orm_o01_message(sender, receiver, data, op)
  elif type == "ORU_R01":
    return generate_hl7_oru_r01_message(sender, receiver, data, op)
  elif type == "ADT_A08":
    return generate_hl7_adt_a08_message(sender, receiver, data, op)
  else:
    raise ValueError("Message Type not suported")
```

Código 1: Geração das mensagens HL7

Como se pode depreender pela figura 1, são utilizados três tipos de mensagens:

- **ORM_O01**, para a requisição, finalização ou cancelamento de pedidos de exames médicos.
- **ORU_R01**, para a comunicação dos relatórios
- **ADT_A08**, para a admissão de pacientes e posterior atualização.

O pedaço de código 2 exemplifica a construção das mensagens **ORM_O01**.

```
m = Message("ORM_O01")

# msh
m.msh.msh_3 = sender
m.msh.msh_4 = sender
m.msh.msh_5 = receiver
m.msh.msh_6 = receiver
m.msh.msh_9 = "ORM^O01"
m.msh.msh_10 = nanoid.generate()
m.msh.msh_11 = "P"

# pid
m.add_group("ORM_O01_PATIENT")
m.ORM_O01_PATIENT.pid.pid_3 = str(data["patient_id"])
m.ORM_O01_PATIENT.pid.pid_5 = data["patient_name"]
m.ORM_O01_PATIENT.pid.pid_11 = data["patient_address"]
m.ORM_O01_PATIENT.pid.pid_13 = data["patient_phone_number"]

# pv1
m.ORM_O01_PATIENT.ORM_O01_PATIENT_VISIT.pv1.pv1_2 = "I"
m.ORM_O01_PATIENT.ORM_O01_PATIENT_VISIT.pv1.pv1_19 = str(data["episode_number"])
```

```
# orc
if op == 1:
  m.ORM_O01_ORDER.orc.orc_1 = "CA"
elif op == 2:
  m.ORM_O01_ORDER.orc.orc_1 = "SC"
  m.ORM_O01_ORDER.orc.orc_5 = "CM"
else:
  m.ORM_O01_ORDER.orc.orc_1 = "NW"
m.ORM_O01_ORDER.ORC.orc_2 = str(data["number"])
m.ORM_O01_ORDER.ORC.orc_3 = str(data["number"])
m.ORM_O01_ORDER.ORC.orc_9 = m.msh.msh_7
```

Código 2: HL7 - ORM_O01

3.3 Comunicação entre serviços

Tal como foi dito no capítulo da arquitetura, foram criados dois *channels* no **Mirth**, Serviço1 e Serviço2, sendo que o Serviço1 lida com mensagens enviadas para o serviço 1 e o Serviço2 lida com as mensagens enviadas para o serviço 2. A figura 4 mostra os dois *channels* criados e a correr no **Mirth** (*Dashboard*).

Status	Name	Rev Δ	Last Deployed	Received	Filtered	Queued	Sent	Errored	Connection
Started	[Default Group]	--	--	91	0	0	71	0	--
Started	Service2	0	2021-03-02 1...	6	0	0	4	0	Idle
Started	Service1	0	2021-03-02 1...	85	0	0	67	0	Idle

Figura 4: Mirth - *Dashboard* (lista de canais existentes e estatísticas associadas)

3.3.1 *Mirth Channels* - Configuração

Cada *channel* foi configurado para usar HL7 como o tipo de dados *inbound* e *outbound* e usou-se um *TCP Listener*, com o modo de transmissão **MLLP**. Ao nível dos *Destinations*, para casa serviço foi criado um *Destination* com um conetor do tipo *Database Driver*. Neste passo, para ser possível aceder facilmente aos campos de interesse das mensagens recebidas foram definidos *transformers* para mapear esses valores para *Channel Mappings* (variáveis do canal).

A figura 5 mostra o espaço de configuração para o *Destination* do *channel* do **Serviço 2**, onde se pode ver tanto as ações que são tomadas, bem como os

Channel Mappings referidos anteriormente, sendo que os que foram definidos por nós são os que começam por letras minúsculas, sendo os restantes mapeamentos criados automaticamente pelo próprio **Mirth** aquando da criação do canal.

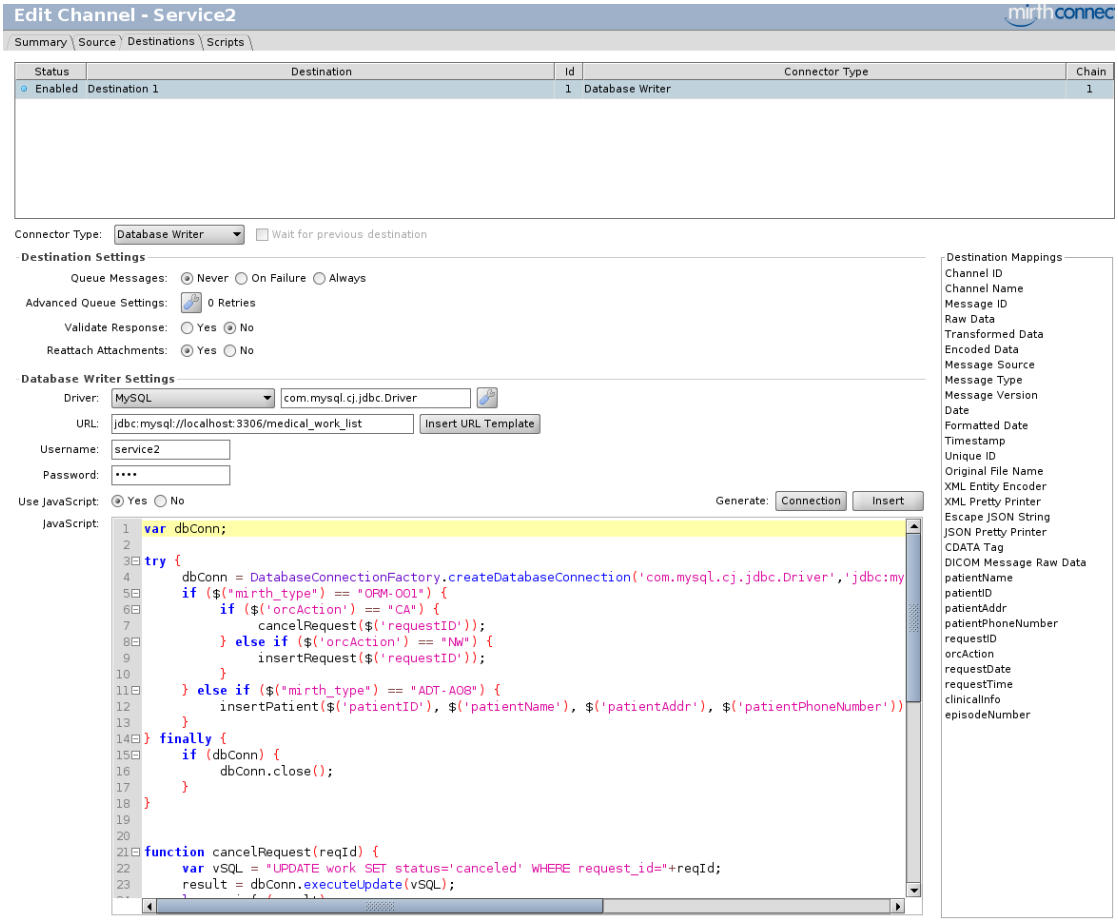

Figura 5: Configuração do *Mirth Connect Channel* para o Serviço 2

4 Resultados

4.1 Apresentação

Com o desenvolvimento deste trabalho foi possível criar um cenário de interoperabilidade semântica em que dois serviços distintos comunicam de forma a completarem um macro-objetivo (a requisição e execução de exames médicos), cada um cumprindo com as suas responsabilidades e fazendo apenas aquilo que foi desenvolvido para fazer, delegando as restantes responsabilidades ao outro serviço.

Como suplemento ao presente relatório e para mais facilmente demonstrar o resultado final do que foi desenvolvido, gravou-se um vídeo que se encontra disponível aqui.

4.2 Discussão

Tendo em conta o resultado atingido importa discutir e analisar o impacto do mesmo. Antes de mais, é importante referir que, de facto, foi possível implementar com sucesso a arquitetura pretendida.

O facto de os dois serviços comunicarem entre si e partilharem a informação relevante sobre os eventos que ocorrem permite que as responsabilidades possam ser localizadas e distribuídas pelos serviços, isto é, faz com que não seja necessário executar todas as ações nos dois serviços, podendo algumas ações serem executadas no serviço 1 e as restantes no serviço 2, pois, eles irão comunicar e partilhar entre si toda a informação necessária de forma a que ambos representem o mesmo estado dos dados.

Esta abordagem possibilita maior eficiência e simplicidade tanto ao nível do sistema, como relativamente aos processos necessários para cumprir a requisição e finalização de um exame médico. Permite, também, tirar um melhor partido dos recursos disponíveis, visto que as funcionalidades estão distribuídas por dois serviços totalmente diferentes. Por fim, esta abordagem distribuída, através da interoperabilidade semântica entre os dois serviços, permite garantir uma maior consistência e resiliência dos dados.

Não sendo tudo vantagens, esta abordagem acarreta o problema de qualquer sistema distribuído: garantir que ambos os serviços funcionam corretamente, pois, eventualmente, a falha de um pode comprometer o correto funcionamento do outro, pelo menos ao nível da atualização dos dados.

Por fim, um aspeto que podia ser considerado para trabalho futuro seria desenvolver uma interface visual (por exemplo, uma página *web*) que permitisse mais facilmente interagir com os serviços desenvolvidos.

5 Conclusão

Com este trabalho foi implementado um mecanismo de interoperabilidade semântica entre dois serviços distintos, através de mensagens *HL7*, usando o *Mirth*. Tendo em conta os objetivos definidos no enunciado, foi possível cumprir com todos os requisitos e ainda implementar a funcionalidade adicional de permitir a admissão de pacientes.

Em suma, a realização deste trabalho permitiu ao grupo alargar os seus conhecimentos relativamente a conceitos expostos nas aulas, pelo que, e tendo em conta o resultado final, considera-se que o resultado do trabalho ultrapassou as expetativas e requisitos definidos.

Referências

[1] *HL7apy - a lightweight Python library to parse, create and handle HL7 v2.x messages*, http://crs4.github.io/hl7apy, Acedido: 01-03-2021.

[2] *python-hl7 - Easy HL7 v2.x Parsing*, https://python-hl7.readthedocs.io/en/latest, Acedido: 03-03-2021.

[3] *Caristix - HL7 Online Documentation*, https://hl7-definition.caristix.com/v2/HL7v2.5, Acedido: 29-02-2021.