Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Programa de Pós-Graduação em Engenharia Elétrica

Allender Vilar de Alencar

Plataforma de Transcodificação para Interoperabilidade de Dados em Saúde

Campina Grande, Paraíba, Brasil

©Allender Vilar de Alencar, Dezembro de 2022

Allender Vilar de Alencar

Plataforma de Transcodificação para Interoperabilidade de Dados em Saúde

Dissertação de mestrado apresentada à Coordenação do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande - Campus de Campina Grande, como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica.

Angelo Perkusich, D.Sc.
Orientador
Danilo Freire de S. Santos, D.Sc.
Orientador

Campina Grande, Paraíba, Brasil
Dezembro de 2022

A368p Alencar, Allender Vilar de.
Plataforma de transcodificação para interoperabilidade de dados em saúde / Allender Vilar de Alencar. – Campina Grande, 2023.
68 f. : il. color.

Dissertação (Mestrado em Engenharia de Elétrica) – Universidade Federal de Campina Grande, Centro de Engenharia de Elétrica e Informática, 2022.
"Orientação: Prof. Dr. Angelo Perkusich, Prof. Dr. Danilo Freire de Souza Santos".
Referências.

1. Processamento da Informação. 2. Interoperabilidade. 3. Infraestrutura. 4. Microsserviços. 5. Malha de Serviço. 6. FHIR. I. Perkusich, Angelo. II. Santos, Danilo Freire de Souza. III. Título.

CDU 621.391(043)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECÁRIA SEVERINA SUELI DA SILVA OLIVEIRA CRB-15/225

Plataforma de Transcodificação para Interoperabilidade de Dados em Saúde

ALLENDER VILAR DE ALENCAR

DISSERTAÇÃO APROVADA EM 19/12/2022

ANGELO PERKUSICH, DEE, UFCG
Orientador(a)

DANILO FREIRE DE SOUZA SANTOS, Dr., UFCG
Orientador(a)

ALEXANDRE JEAN RENÉ SERRES, D.Sc., UFCG
Examinador(a)

DALTON CEZANE GOMES VALADARES, Dr., IFPE
Examinador(a)

CAMPINA GRANDE - PB

Dedicatória

Dedico a todos aqueles que já foram salvos, ao menos uma vez, pela ciência.

Agradecimentos

Agradeço primeiramente aos cientistas do passado, porque sem os seus estudos sobre diversas áreas até mesmo a de anatomia, possivelmente, não estaria aqui para realizar este trabalho.
À minha mãe, Ana, aos meus irmãos, Alluska, Aryadne e Aristótenes V. Filho, pelo apoio e ajuda em grande parte da minha vida, e ao meu pai. A todos os membros da minha família, seja de sangue, ou não, e em especial, à minha tia e meus avós por parte de mãe que deram bastante suporte ao longo da vida.
Aos meus amigos que desferem as críticas mais incisivas, ajudando-me assim, a ter, não somente uma nova visão sobre as coisas, mas também, formas de como me aprimorar e me entender.
Ao professores do curso de Engenharia Elétrica que cada qual com sua convicção luta firme e com muito amor ao curso para aprimorá-lo e ajudar os nossos queridos estudantes. Em especial, aos meu oriantadores, Angelo e Danilo, por darem o suporte necessário para realização deste trabalho.
Também agradeço ao CNPq, à CAPES ou a quem de direito que tenha proporcionado o suporte financeiro para viabilizar a realização de seu trabalho.

Resumo

Nesta dissertação é apresentada uma plataforma baseada em microsserviços, utilizando malha de serviço, que transcodifica dados entre padrões de saúde, utilizando o padrão de saúde *Fast Healthcare Interoperability Resources* como base, com objetivo de prover interoperabilidade. O *Fast Healthcare Interoperability Resources* é utilizado como padrão base, pois é um padrão com tecnologias Web e apresenta tendência de ser utilizado em boa parte da indústria. Os resultados obtidos evidenciam que o uso do *Fast Healthcare Interoperability Resources* como padrão central de interoperabilidade em um sistema de transcodificação torna possível conectar diferentes sistemas entre si e prover interoperabilidade. Os resultados obtidos nos experimentos demostraram que o sistema proposto é escalável para cenários de alta demanda, como pandemias, e passível de ser reconfigurável sem a interrupção da execução da plataforma. Essas características permitem manter a qualidade de serviço, aumentar a disponibilidade e realizar reconfigurações sem reiniciar o sistema. Por fim, foi possível demonstrar que é possível encapsular funções de transcodificação de modo independente, permitindo que essas funções possam se beneficiar das características do modelo arquitetural de microsserviços.

Palavras-chave: Interoperabilidade, Infraestrutura, Microsserviços, Malha de serviço, FHIR.

Abstract

This dissertation presents a platform based on microservices using a service mesh, which transcodes data between health standards, using the Fast Healthcare Interoperability Resources as a central standard, Because it is a standard with web technologies and is widely adopted by the industry. The results show that using Fast Healthcare Interoperability Resources as a base interoperability standard in a transcoding system makes it possible to connect different systems and provide interoperability. The results obtained from the experiments showed that the proposed system is scalable for scenarios of high demand, such as pandemic, and capable of being reconfigurable without interrupting the platform's execution. Therefore, the solution allows for maintaining service quality, increasing reliability, and carrying out reconfiguration without restarting the system. It was possible to demonstrate that it is possible to encapsulate independent transcoders functions, allowing these functions to take advantage of characteristics of the microservices architectural model.

Keywords: Interoperability, Infrastructure, Microservices, Service Mesh, FHIR.

Sumário

Lista de símbolos e abreviaturas

AAA *Authentication, Authorization and Audit (Autenticação, Autorização e Auditoria)* 24

API Interface de Programação de Aplicação (*Application Programming Interface*) 14

CCDA Arquitetura de documento clínico consolidada (*Consolidated Clinical Document Architecture*) 17

CDA Arquitetura de documento clínico (*Clinical Document Architecture*) 17

DSRM Metodologia de Pesquisa em Design Science (*Design Science Reseach Methodology*) 7

EHR Prontuário Eletrônico do Paciente (*Eletronic Health Record*) 1

FHIR Recursos Rápidos de Interoperabilidade de Assistência Médica (*Fast Healthcare Interoperability Resources*) 3

HIMSS Sociedade de Sistemas de Informação e Gestão em Saúde (*Healthcare Information and Management Systems Society)* 12

HL7 *Health Level Seven* 3

HTTP Protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol*) 3

HTTP Protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol*) 15

ICE Ambiente Clínico Integrado (*Integrated Clinical Environment*) 27

IGaaS Gateway IoT como Serviço(*IoT Gateway as a Service*) 14

IoHT Internet das Coisas de Saúde(*Internet of Healthcare Things*) 22

IoT Internet das Coisas (*Internet of Things*) 1

PHR Prontuário de Saúde do Paciente (*Patient Health Record*) 1

PN Navegação do Paciente (*Patient Navigation*) 6

QoS Qualidade de Serviço (*Quality of Service*) 30

REST Transferência de Estado Representacional (*Representational State Transfer*) 15

RNDS *Rede Nacional de Dados em Saúde* 4

TaaS Transcodificação como um Serviço (*Transcoding as a service*) 14

TLS *Transport Layer Security* (Segurança da Camada de Transporte) 24

UUID Identificador Único Universal (*Universal Unique Identifier*) 33

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O setor da saúde é um dos mais importantes da sociedade, o que acaba atraindo interesse e investimentos para o desenvolvimento de novas tecnologias e técnicas que o aprimorem. Nesse quesito, com o avanço da Internet das Coisas (do inglês *Internet of Things* - IoT), os dispositivos médicos com essa tecnologia têm um papel fundamental no monitoramento de quadros clínicos, por exemplo, permite o acompanhamento de medições de temperatura e pressão do paciente. Além disso, através da integração com sistemas de informação, é possível criar alertas em casos de medições que não estão em valores aceitáveis, como também batimentos cardíacos lentos [1]. Com o passar do tempo, houve um aumento no número de pacientes que precisam desse tipo de observação, o que gerou o problema da falta de especialistas, qualificados na saúde, para realizarem as medições dos pacientes [2]. Assim, a utilização dos dispositivos IoT se torna uma solução viável, dado que não é necessário um profissional da saúde realizar as medições manualmente, e em casos menos críticos, onde o paciente não corre risco de morte, essa observação do quado clínico pode ser feita remotamente, isto é, o paciente não precisa ficar no hospital [2].

Monitorar, realizar exames, entre outros procedimentos, acabam por gerar cada vez mais dados armazenados nos bancos de dados de cada provedor de serviços médicos. Com isso, questiona-se até que ponto essa quantidade de dados gerados pelas tecnologias pode colaborar para avanços na medicina? Nesse sentido, faz-se necessário explorar mecanismos que possam viabilizar o acesso a esses dados de saúde, de forma mais ampla e colaborativa, para aplicações que possam contribuir para o setor da saúde em âmbito regional e até global.

Segundo Saripalle *et al.* [3], é definido que o Prontuário de Saúde do Paciente (do inglês *Patient Health Record* - PHR) é o histórico de saúde do paciente em um formato de dados padronizado, o qual o mesmo tem acesso e pode compartilhar suas informações com pessoas autorizadas em um ambiente seguro. O Prontuário Eletrônico do Paciente (do inglês *Eletronic Health Record* - EHR), que armazena o mesmo tipo de informação que o PHR, é feito e atualizado por profissionais da área. Dessa forma, existem três ligações possíveis entre o PHR e o EHR:

- *individual*: permite o paciente gravar seus dados médicos no PHR, mas não conecta a nenhum EHR;
- *amarrado*: conecta e sincroniza o PHR com um banco EHR;
- *integrado*: conecta e sincroniza o PHR com vários bancos EHR.

Com isso, especialistas apontam que a forma integrada com conexão a vários EHR beneficia o PHR, pois é possível em um único sistema ter a análise horizontal do paciente, contendo consultas, diagnósticos, histórico de remédios, etc. [3].

Entretanto, dados de saúde, ou seja, os dados que são armazenados no PHR e EHR, só são úteis quando são possíveis de serem transformados em informações significativas, isto é, as informações poderem ser trocadas e utilizadas por outros serviços de saúde, e isso requer interoperabilidade entre os sistemas médicos, como hospitais, por exemplo. Dessa forma, existem quatro áreas principais que podem ser beneficiadas pela interoperabilidade médica [4]:

- inteligência artificial e *big data*: com o aumento do banco de dados disponível devido à interoperabilidade, é possível validar resultados e análises e utilizar em algoritmos de inteligência artificial;
- comunicação médica: possibilita a comunicação entre diferentes sistemas, assim é possível trocar informações tornando os diagnósticos mais claros e concisos, pois é possível ter acesso ao histórico médico de forma mais ampla;
- pesquisa: neste ponto, gera uma melhora no estudo de uma determinada doença pela vasta quantidade de diagnósticos de análises, ou até mesmo doenças raras, também

traz a possibilidade de criar hipóteses através da análise de dados, e inteligência artificial;

- cooperação internacional: torna possível realizar rastreio de problemas de saúde pública mundial, principalmente em controle de infecções.

Com isso fica clara a grande importância da interoperabilidade na saúde. Entretanto, atualmente a maioria desses dados está isolada em banco de dados de sistemas que não conseguem realizar essa comunicação entre si [4].

Com o exposto, identifica-se duas possibilidades para mitigar o problema apresentado: (i) a necessidade de adoção de um padrão central a ser utilizado por todos os sistemas para prover a interoperabilidade entre os sistemas de saúde; (ii) adoção de transcodificadores capazes de realizar a comunicação entre os sistemas e reorganizar o formato dos dados provendo essa interoperabilidade. A primeira solução é mais complexa, pois envolve alterar todos os sistemas existentes para utilizar o mesmo padrão, o que pode acarretar atualizações em banco de dados e interfaces de comunicação. Além disso, esse tipo de solução pode apresentar um custo elevado, dificultando a aceitação da mesma [5]. Na segunda, os transcodificadores, por outro lado, não necessitam de maiores alterações nos sistemas. Sendo assim, essa solução se apresenta possivelmente como a mais viável para interligação entre padrões de interoperabilidade e os sistemas legados, que são os sistemas mais antigos que ainda operam.

Nesse contexto, o padrão *Fast Healthcare Interoperability Resources* (FHIR) foi criado pela organização de padrões de saúde *Health Level Seven International* (HL7) [6], tendo duas funções principais: a primeira é de definir um padrão para todos os tipos de dados médicos que são armazenados; a segunda é a criação de recursos universais normalmente implementados com tecnologias Web como o protocolo de comunicação *Hypertext Transfer Protocol* (HTTP). Com isso é possível ter um banco de dados unificado e compartilhado, tornando os dados mais acessíveis [7]. Um exemplo de recurso é o de paciente, o qual contém as informações do paciente como nome, endereço e telefone.

Devido às vantagens do FHIR de ser facilmente acessível via Web e estar sendo cada vez mais implementado, esse padrão apresenta potencial em ser o padrão base dos transcodificadores. Por exemplo, podem ser implementados transcodificadores entre um padrão A e

o FHIR, e entre um padrão B e o FHIR, desse forma, é possível transcodificar o padrão A no padrão B, basta apenas transcodificar o padrão A em FHIR, e depois de FHIR para o padrão B, e vice-versa. Isso reduz a implementação, pois não é necessário implementar o transcodificador do padrão A para o padrão B.

No Brasil, a Rede Nacional de Dados em Saúde (RNDS) é uma solução em nuvem que está sendo implementada, que se baseia no padrão de interoperabilidade HL7 FHIR r4. Entretanto, esta solução não contempla toda a interoperabilidade com os sistemas legados já existentes [8]. Dessa forma, a transcodificação desses dados pode ser um mecanismo facilitador para proporcionar a comunicação entre sistemas de provedores de saúde e também com a RNDS, sendo uma solução que integra os sistemas entre si, mas não cria a necessidade de realizar mudanças dos sistemas existentes [5].

Provedores de saúde podem necessitar se comunicar dinamicamente a depender de diferentes situações. Por exemplo, em um cenário de epidemia em uma localidade, os serviços de saúde de uma região precisarão trocar informações em maior escala durante curtos intervalos de tempo. Portanto, além do desafio de prover interoperabilidade entre esses sistemas, existem também os desafios relativos ao modelo arquitetural dos sistemas de comunicação e informação em saúde, de modo que eles consigam satisfazer requisitos como escalabilidade e confiabilidade, e ainda satisfazerem requisitos de qualidade em serviços de saúde, como os apresentados no padrão IEEE 11073:00101 [9]. O padrão IEEE 11073 define modelos e protocolos para que dispositivos de saúde possam coletar e compartilhar dados [9]. Alguns requisitos de qualidade de serviço do IEEE 11073 são apresentados no documento IEEE 11073:00101, dos quais destacamos:

- tempo máximo para comunicação de um alarme, que pode indicar algum acontecimento importante das medições dos sinais vitais do paciente, é de 3 segundos;
- a busca por prontuários médicos que deve levar no máximo 5 segundos.

Os sistemas médicos realizam uma grande quantidade de trocas de dados, chegando até mil requisições por segundo, e processamentos, como em monitoramentos médicos em tempo real [10]. Nesse cenário, a computação de borda é vista como uma possível solução para os transcodificadores, pois os transcodificadores terão que processar dados de múltiplos

sistemas médicos, podendo atingir milhares de requisições [2]. No mesmo cenário de epidemia localizada apresentado anteriormente, soluções como computação na borda ajudam a manter a Qualidade de Serviço de saúde, por exemplo, o apresentado no padrão IEEE 11073:00101 [9].

O padrão de projeto arquitetural de microsserviços é um estilo de implementação de serviços que os tornam, geralmente, independentes entre si. Essa característica de desacoplamento faz com que seja possível escalar horizontalmente os microsserviços [11]. Dessa forma, microsserviços são mais flexíveis e escaláveis do que estilos arquiteturais que não apresentam essa independência entre serviços. No caso dos transcodificadores para saúde, os aspectos de escalabilidade e flexibilidade são importantes, devido à necessidade de aumentar ou diminuir a quantidade de transcodificadores a depender da quantidade de requisições sendo realizadas aos mesmos [10], ou seja, manutenção da qualidade de serviço. A característica de escalabilidade e desacoplamento entre serviços aumenta a confiabilidade dos mesmos, tornando o sistema mais tolerante a falha [12].

Considerando o cenário apresentado no trabalho de Bettoni *et al.* [13] para a situação de Navegação do Paciente (do inglês *Patient Navigation - PN*) e agendamentos (do inglês *Appoitments*), foi necessária a utilização da arquitetura de microsserviços para implementação somente de um servidor, pois esse modelo de arquitetura fornece isolação entre serviços para corrigir problemas de persistência de dados e diminui a complexidade de escalar a aplicação. Devido à possibilidade dos transcodificadores tratarem os dados de muitos servidores e em um contexto mais amplo que o do trabalho citado, além de inserir um tempo de atraso entre a comunicação dos servidores, os mesmos precisam ser ágeis. Principalmente em cenários de maior demanda de troca de dados como, por exemplo, pandemias, a arquitetura dos transcodificadores nesses cenários também necessitará da flexibilidade, escalabilidade e desacoplamento que os microsserviços fornecem.

Com isso, este trabalho é motivado pelo seguinte problema de engenharia: dado que existem padrões de saúde que não conseguem trocar informações entre si, a interoperabilidade ainda é um problema pela falta de capacidade desses sistemas trocarem informações e as utilizarem da forma correta, ou seja, comunicar, entender, salvar e processar os dados. Dessa forma, identifica-se como desafio a necessidade de transcodificar padrões de saúde para prover interoperabilidade entre sistemas de saúde distintos, considerando os requisitos:

- cumprir requisitos de qualidade de serviço de saúde, como tempo de resposta;
- o sistema deve ter uma forma de aumentar a disponibilidade, ou seja, diminuir a probabilidade de estar fora de operação;
- alterar os padrões que estão sendo transcodificados sem impactar no funcionamento da plataforma, isto é, manter os serviços em funcionamento mesmo sendo feitas configurações;
- flexibilidade de inserir novos provedores de saúde na comunicação, mantendo a segurança e funcionamento do sistema.

1.1 Objetivo Geral

Com base nesse cenário, este trabalho tem por objetivo projetar e desenvolver uma arquitetura de um sistema de transcodificadores em saúde, onde as funções de transcodificação são independentes entre si, satisfazendo os seguintes requisitos:

- escalabilidade em situações de alta demanda de serviços de transcodificação, para manter o tempo de resposta dentro dos limites da qualidade de serviço de saúde;
- flexibilidade de reconfiguração para adição, remoção ou evolução de transcodificadores de modo independente;
- disponibilidade do serviço de transcodificação para garantir os indicadores de qualidade de serviço de saúde.

Considerando esses requisitos e cenário, o modelo arquitetural para o sistema de transcodificação desse trabalho baseia-se em uma arquitetura de microsserviços com uma malha de serviço, como será detalhado e justificado nas próximas seções e capítulos.

1.2 Objetivos Específicos

Estão sendo propostos os seguintes objetivos específicos no desenvolvimento do trabalho:

- investigar padrões de dados de saúde e seus protocolos, ferramentas existentes para interoperabilidade de dados médicos, e soluções arquiteturais existentes;
- investigar a viabilidade do uso de uma arquitetura de microsserviços para sistemas de transcodificação, avaliando os requisitos de qualidade de serviços apresentados no padrão IEEE 11073:00101 [9];
- implementar prova de conceito do modelo de transcodificação/mapeamento, em microsserviços, entre HL7 v2 e o FHIR, entre C-CDA e o FHIR, e comparar com as ferramentas existentes;
- validar o funcionamento do sistema proposto comparando com ferramentas existentes de transcodificação, por requisitos funcionais e não funcionais;
- verificar funcionamento do sistema, escalando a aplicação, em relação ao tempo de resposta, observando se cumpre os requisitos de qualidade de serviço de sistemas em saúde.

1.3 Metodologia

De acordo com Wieringa *et al.* [14], *Design Science Reseach Methodology* (DSRM) é uma metodologia adotada para solucionar problemas de engenharia de software. Essa metodologia é definida, de forma simplficada, em um *Design Cycle* conforme apresentado na Figura 1.1. Esse ciclo é definido pelas seguintes etapas [14]:

- *Avaliação da Implementação/Investigação do Problema*: caso nessa parte do ciclo já tenha implementação, então é necessário realizar uma avaliação se a solução resolve o problema, caso não haja solução ainda do problema, deve-se investigar o problema e suas causas;
- *Design da Solução do Problema*: nesta etapa é feita a especificação dos requisitos, e com os requisitos são avaliadas as possibilidades de utilizar soluções existentes ou de produzir novas;

- *Validação da Solução*: neste ponto é realizada uma avaliação se o design atende aos requisitos e quais são os seus impactos na solução;
- *Implementação da Solução*: nesta etapa é realizada a solução do problema, considerando as outras etapas já realizadas;

Figura 1.1: Ciclo de design da metodologia.

Fonte: Adaptado de [14].

Neste trabalho, o método utilizado foi o *Design Science Reseach Methodology*, onde foram realizadas pesquisas sobre o tema de interoperabilidade de dados de saúde. Com isso, foi verificado que existe o problema de padrões de saúde diferentes não conseguirem se comunicar entre si, mesmo havendo padrões de saúde desenvolvidos para essa finalidade. Para resolver esse problema, a solução foi dividida em duas etapas: pesquisar e propor solução arquitetural, e validar se a solução arquitetural corresponde aos requisitos de qualidade de serviço de saúde; implementar a solução e verificar resultados. Assim, na primeira etapa é proposta uma nova arquitetura para os transcodificadores, sendo ela uma arquitetura em microsserviços. Essa arquitetura tem as seguintes melhorias:

- flexibilidade de seleção de transcodificares: inserir, remover e atualizar transcodificadores sem impactar o funcionamento dos outros que estão em execução é um ponto importante para manutenção da comunicação;
- escalabilidade de transcodificadores: aumentar o uso de recursos computacionais para suprir um aumento significativo na quantidade de requisições recebidas;
- flexibilidade de inserir novos provedores de saúde: a plataforma deve conseguir inserir novos provedores em sua infraestrutura de forma fácil, e mantendo a segurança;

- confiabilidade dos serviços: os serviços devem ter maior confiabilidade, diminuindo a probabilidade de serviços estarem indisponíveis por falhas.

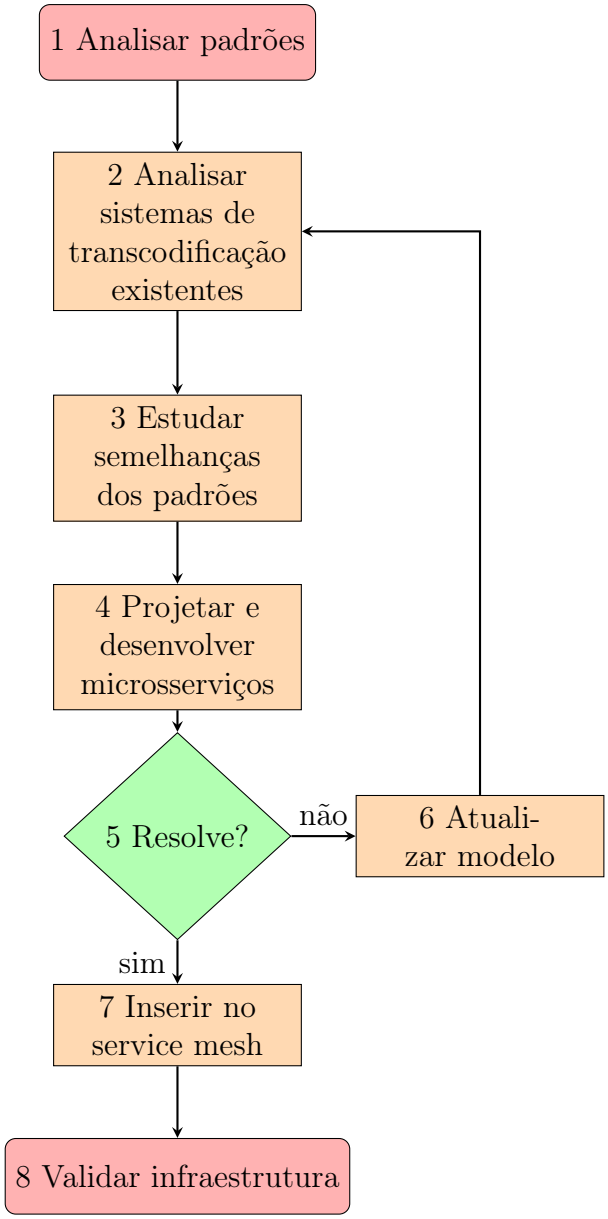
Para a segunda etapa de solução do problema, o fluxograma seguido para essa etapa é apresentado na Figura 1.2, com isso temos:

- (1) *Analisar padrões*: foi realizado um estudo dos padrões, como são estruturados e como transcodificar os padrões para o FHIR;
- (2) *Analisar sistemas de transcodificação existentes*: esse passo é importante para entender como os sistemas existentes funcionam e como é possível realizar transcodificação;
- (3) *Estudar semelhanças dos padrões*: entender quais são as semelhanças dos campos de cada padrão para mapeá-los diretamente, e os que diferem serem tratados para serem adicionados ou ignorados, a depender do caso;
- (4) *Projetar e desenvolver os microsserviços*: com as informações coletadas até o passo (3), os microsserviços de teste foram implementados;
- (5) *Resolve?*: essa decisão é analisada a partir de testes de transcodificação comparando com a plataforma desenvolvida pela Microsoft [15], com isso é levado ao passo (6) ou (7);
- (6) *Atualizar modelo*: caso o passo (5) obtenha resultado negativo, então a execução volta ao passo (2) para analisar os sistemas já existentes para ajustar a implementação;
- (7) *Inserir no service mesh*: com os microsserviços implementados podem ser inseridos na *service mesh* para ter as vantagens dessa camada;
- (8) *Validar infraestrutura*: é realizada a validação do funcionamento do sistema por meio de testes.

1.4 Contribuições

As principais contribuições desta dissertação são:

Figura 1.2: Fluxograma da metodologia.



Fonte: elaborada pelo autor.

- definir uma arquitetura geral, em microsserviços, para prover interoperabilidade;
- criar um modelo de implementação de microsserviços para inserção na arquitetura;
- construir uma arquitetura de simples configuração de serviços de transcodificação e provedores de saúde;
- implementar um sistema de referência para ser compartilhado para a comunidade científica.

1.5 Organização do Trabalho

Este documento está sequenciado pela divisão dos seguintes capítulos:

- **Capítulo 1 - Introdução**: neste primeiro capítulo foram apresentados as motivações do trabalho, metodologia, contribuição e os objetivos;
- **Capítulo 2 - Fundamentação Teórica**: apresenta-se uma revisão bibliográfica e de tecnologias, onde são abordados os principais trabalhos realizados em interoperabilidade, tais como implementações de sistemas baseados em padrões interoperáveis, soluções e as tecnologias utilizadas nesta dissertação;
- **Capítulo 3 - Modelo Arquitetural**: são apresentadas as premissas e a proposta da solução arquitetural;
- **Capítulo 4 - Implementação e Resultados Experimentais**: são apresentados a plataforma e os resultados experimentais;
- **Capítulo 5 - Considerações Finais**: são apresentadas as conclusões e a proposta para trabalho futuro.

Capítulo 2

Fundamentação Teórica

Dado o contexto da problemática, neste capítulo apresenta-se a fundamentação teórica acerca de interoperabilidade e transcodificação para sistemas médicos. Com isso, foram realizadas revisões nos temas de: sistemas de computação na borda focada em cuidados de saúde e interoperabilidade para sistemas médicos. A partir dessas revisões foram selecionados os artigos mais relevantes para compor o embasamento desta dissertação.

Este capítulo está dividido sequencialmente da seguinte forma: são apresentados inicialmente a definição dos conceitos interoperabilidade, transcodificação, *Application Programming Interface*, padrão FHIR, microsserviços e malha de serviço, e em seguida é apresentada a revisão bibliográfica acerca de interoperabilidade e as implementações que tentam provê-la.

2.1 Interoperabilidade

Interoperabilidade dos dados em formato EHR é definida, pelo *Healthcare Information and Management Systems Society* (HIMSS), como a habilidade de duas ou mais aplicações conseguirem se comunicar de forma efetiva sem comprometer o conteúdo do EHR, independente de qual seja o padrão utilizado. Com isso é possível compartilhar EHR entre hospistais, clínicas, laboratórios e outras unidades de pesquisa, facilitando a troca de dados pelos usuários e especialistas [16].

Interoperabilidade de dados médicos é dividida em quatro níveis e cinco categorias [17, 18], sendo os níveis:

- técnica ou fundacional: estabelece inter-conectividade em um sistema ou aplicação para comunicação segura dos dados;
- sintática ou estrutural: define o formato, a sintaxe e a organização dos dados para a troca dos mesmos;
- semântica: é a capacidade de dois ou mais sistemas "compreenderem" o conteúdo dos dados baseados nas terminologias utilizadas, por exemplo, uma palavra pode ter mais de um significado;
- organizacional: esse ponto considera governo, políticas, considerações sociais, legais e organizacionais para a troca de dados entre seus agentes e sistemas, provendo segurança da informação, consentimento e confiança dos dados.

A HIMSS [18] define que as categorias são:

- vocabulário ou terminologia: define as terminologias de forma concisa, evitando representações ambíguas, tornando a comunicação eficaz e mais compreensível;
- conteúdo: define a estrutura e organização dos dados eletrônicos nas mensagens ou documentos, essa categoria também inclui a definição de conjuntos de dados para tipos de mensagens específicas;
- transporte: formata as mensagens trocadas entre sistemas de computadores, arquitetura de documentos, modelos clínicos, interface de usuário e ligação de dados de usuários;
- privacidade e segurança: lida diretamente com proteger os direitos do indivíduo, ou das organizações, para determinar o quê, quando, por quem e para qual propósito os dados de saúde estão sendo coletados, acessados, usados ou divulgados;
- identificadores: utiliza padrões de identificadores para identificar pacientes ou provedores de forma exclusiva.

2.2 Transcodificação

A transcodificação é o processamento para mudar o formato dos dados de um padrão com uma certa finalidade, seja a de compressão para uma transmissão mais enxuta, mudança de padrão para interoperabilidade entre sistemas, etc. Esses algoritmos de transcodificação são complexos, mas apresentam um papel fundamental nas comunicações [19].

A transcodificação pode ser oferecida como um serviço (TaaS), tendo assim uma redução da complexidade para o usuário na definição de tarefas para transcodificar, executá-las e realizar o manejamento das saídas dos transcodificadores pelos protocolos específicos [20]. Esse conceito também é conhecido por *IoT Gateway as a Service* (IGaaS), que é a possibilidade de escalabilidade dos serviços de transcodificação, ou mapeamento, ou *gateway* [21]. Essas transcodificações, TaaS e IGaaS têm a vantagem de melhor utilização da infraestrutura, podendo ser implementadas em hardwares mais limitados que servidores tendo um custo menor e pouco gasto energético.

2.3 Interface de Programação de Aplicação

Interface de Programação de Aplicação (do inglês *Application Programming Interface* - API) é definida como interface de programação de aplicações, tecnicamente é a forma como aplicações de *software* realizam a comunicação, trocam dados e realizam chamadas de funcionalidades específicas que são realizadas utilizando definições concisas e protocolos [22]. Com isso é possível desenvolver aplicações e integrá-las mais facilmente. Por exemplo, sistemas operacionais utilizam APIs para simplificar o trabalho de desenvolver aplicações para os mesmos, definindo funções com o papel de facilitar o desenvolvimento e integração como já citado, no caso de aplicações gráficas, por exemplo, existem as interfaces OpenGL, Vulkan e DirectX [23].

Como as APIs funcionam como mediadores, elas têm papel fundamental em evolução de sistemas, pois é possível manter os softwares que realizam as requisições utilizando as APIs e trocá-las desde que sigam as mesmas regras de comunicação. Dessa forma, é possível, por exemplo, trocar um servidor FHIR por uma API similar, ou seja sua interface de requisições continua a mesmo, mas no lugar de executar as funcionalidades para manusear os dados no

servidor, essas requisições podem ser mapeadas para outros sistemas ou APIs.

2.3.1 *REpresentational State Transfer*

REpresentational State Transfer REST não é um protocolo, mas sim um estilo arquitetural, um padrão de design (*Design Pattern*) criado por Roy Fielding em sua tese de doutorado. Além disso, para uma arquitetura ser considerada RESTful tem que seguir os cinco critérios[24]:

- cliente-servidor : seguir arquitetura cliente-servidor, onde o cliente é responsável por realizar as chamadas *Hypertext Transfer Protocol* (HTTP);
- sem estado : comunicação *Stateless* implica que toda a informação necessária é contida na requisição, todas as requisições são separadas e desconexas;
- *Caching* : Armazena dados em cache para diminuir o uso do servidor;
- sistema em camadas : sistema em camadas, onde o usuário não consegue distinguir a camada de acesso;
- interface uniforme : como é mostrado mais abaixo, é utilizada uma descrição bem definida da API.

O protocolo HTTP define alguns métodos, a seguir estão os mais utilizados ao realizar requisições em *endpoints*, por exemplo os métodos podem ser utilizados para realizar mudanças nos usuários:

- GET (Recebe informações de usuários, caso tenha permissão)
- PUT (Altera informações do usuário, caso tenha permissão)
- POST (Cria novo usuário)
- DELETE (Deleta usuário, caso tenha permissão)

Esse exemplo de usuários é bem comum nos sistemas, pois cada usuário vai ter seu *login*, senha e outros campos necessários, então é possível que os sistemas consigam via essa API REST alterar, saber se o usuário existe, criar e deletar.

2.4 Padrões de Saúde

Nesta seção são apresentados os três padrões de saúde que foram utilizados no trabalho: HL7 v2, FHIR e CCDA. Eles são apresentados de forma resumida, somente para se compreender a estrutura e ser possível entender as informações contidas nas mensagens.

2.4.1 HL7 v2

HL7 v2 é um padrão de dados de saúde baseado em mensagens de texto [25]. Essa mensagem do HL7 v2 é composta por grupos de segmentos ordenados que são as linhas da Lista 2.1. Cada segmento desse é composto por componentes que são divididos por barras verticais "|" e subcomponentes que são divididos por acento circunflexo "^" [26].

```
1 MSH|^~\&|NISTEHRAPP|NISTEHRFAC|NISTIISAPP|NISTIISFAC
    |20150624084727.655-0500||VXU^V04^VXU_V04|NIST-IZ-AD-2.1
    _Send_V04_Z22|P|2.5.1|||ER|AL|||||Z22^CDCPHINVS|NISTEHRFAC|
    NISTIISFAC
2 PID|1||90012^^^NIST-MPI-1^MR||Wong^Elise^^^^L||19830615|F||2028-9^
    Asian^CDCREC|9200 Wellington Trail^^Bozeman^MT^59715^USA^P||^PRN^
    PH^^^406^5557896~^NET^^Elise.Wong@isp.com|||||||||2186-5^Not
    Hispanic or Latino^CDCREC||N|1|||||N
3 OBX|3|CE|69764-9^Document Type^LN|3|25308869830002881117041 1^Tetanus
    /Diphtheria (Td) Vaccine VIS^cdcgs1vis||||||F|||20150624
```

Lista 2.1: Exemplo de Mensagem HL7 v2

A mensagem apresentada na Lista 2.1 é descrita abaixo por linhas.

Cabeçalho, linha 1:

- VXU∧V04: Atualização não solicitada de prontuário de vacinação;
- NISTEHRAPP: Nome da fonte de onde está sendo enviada a mensagem;
- NISTEHRFAC: Nome do destino para onde está sendo enviada a mensagem;
- 20150624084727.655-0500: mensagem enviada no dia 24 de junho de 2015 às 08h47min27s, -5 UTC;

- 2.5.1: versão do hl7 utilizada.

Informações do paciente, linha 2:

- 1: indica ocorrência na mensagem, sendo 1 a primeira ocorrência, caso houvesse outro pid após esse, deveria conter valor 2;
- 90012∧ ∧ ∧NIST-MPI-1∧MR: 90012 é o valor do identificador do paciente na lista do MR(*Medical Record Number*), no caso de transcodificadores, pode-se utilizar de um identificador universal como o CPF;
- Wong∧ELise: Elise Wong é o nome do paciente;
- 19830615: data de nascimento é 15 de junho de 1983;
- F: gênero feminino;
- 9200 Wellington Trail∧∧Bozeman∧MT∧59715∧USA∧P: endereço da paciente é rua 9200 Welligton Trail, cidade Bozeman, estado MT, país Estados Unidos da America, Código postal 59715.

Observação, linhas 3:

- 25308869830001288117041l: indica o código de barras do tipo vis, esse é indicado por cdcg1vis;
- Tetanus/Diphtheria (Td) Vaccine: vacina para tetano e difteria aplicada na paciente;
- 20150624: no dia 24 de junho de 2015 a vacina foi aplicada na paciente.

2.4.2 CCDA

Arquitetura de documento clínico (CDA) é o formato CDA consolidado. Essa arquitetura é um padrão de marcação de documento apresentado na Lista 2.2 [27].

```
1 <author>
2         <time value="20050329224411+0500"/>
3         <assignedAuthor>
```

```
4                 <id extension="KP00017" root="2.16.840.1.113883.19.5
                     "/>
5                 <addr>
6                         <streetAddressLine>21 North Ave.</
                             streetAddressLine>
7                         <city>Burlington</city>
8                         <state>MA</state>
9                         <postalCode>02368</postalCode>
10                        <country>USA</country>
11                </addr>
12                <telecom use="WP" value="tel:(555)555-1003"/>
13                <assignedPerson>
14                        <name>
15                                <given>Henry</given>
16                                <family>Seven</family>
17                        </name>
18                </assignedPerson>
19        </assignedAuthor>
20</author>
21<custodian>
22        <assignedCustodian>
23                <representedCustodianOrganization>
24                        <id root="2.16.840.1.113883.19.5"/>
25                        <name>Good Health Clinic</name>
26                        <telecom value="tel:(555)555-1212" use="WP"
                             />
27                        <addr use="WP">
28                                <streetAddressLine>17 Daws Rd.</
                                     streetAddressLine>
29                                <city>Blue Bell</city>
30                                <state>MA</state>
31                                <postalCode>02368</postalCode>
```

```
32                                    <country>USA</country>
33                              </addr>
34                        </representedCustodianOrganization>
35                </assignedCustodian>
36 </custodian>
```

Lista 2.2: Exemplo de Mensagem CCDA

A mensagem apresentada na Lista 2.2 é descrita abaixo por *author* e *custodian*.

- *<time value="20050329224411+0500»*: data e horário do envio, essa mensagem foi enviada dia 20 de maio de 2005 às 22h44min11s +5 UTC;
- <assignedAuthor>: informações sobre o autor

  <id root="2.16.840.1.113883.19.5»: identificador do autor;

  <addr>: o endereço do autor na mensagem é rua 21 North Ave., cidade Burlington, estado MA, código postal 02368, Estados Unidos da América;

  <Telecom use="WP"value="tel:(555)555-1003»: Número de telefone de trabalho e pessoal;

  <assignedPerson>: nome do autor é Henry Seven;

- <assignedCustodian> <representedCustodianOrganization>: indica a organização de custódia associada à informação:

  <id root="2.16.840.1.113883.19.5"/>: identificador da organização;

  <name>: nome da organização é Good Health Clinic;

  <telecom value="tel:(555)555-1212"use="WP"/>: telefone de trabalgo e pessoal;

  <addr use="WP»: o endereço da clínica é rua 17 Daws Rd., cidade Blue Bell, estado MA, código postal 02368, Estados Unidos da América;

2.4.3 *Fast Healthcare Interoperability Resources*

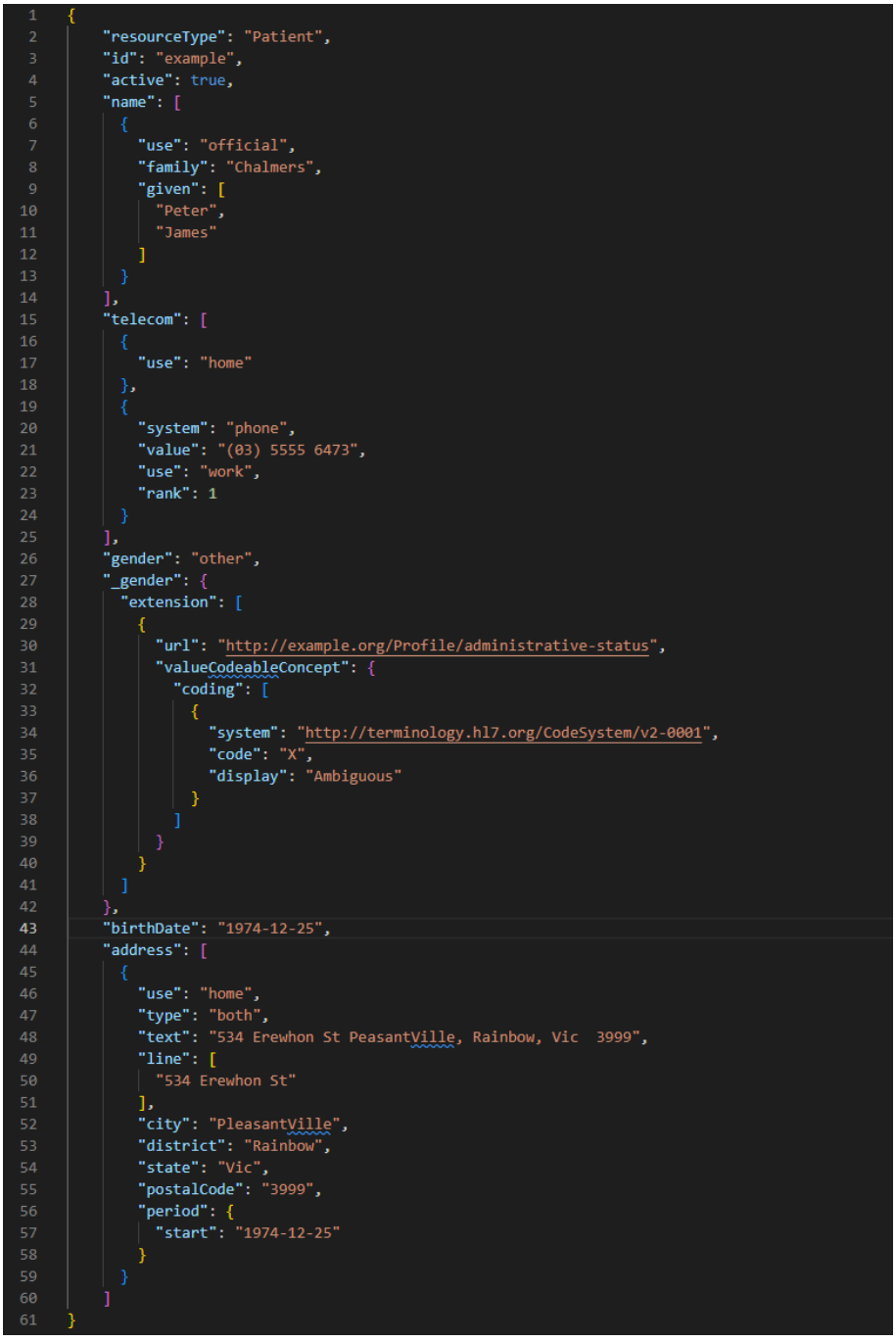
HL7 FHIR é um dos padrões mais recentes para prover interoperabilidade de sistemas de saúde. Baseado nos recursos dos antigos HL7 v2[1], HL7 v3[2] e HL7 CDA[3] traz uma versão utilizando as últimas tecnologias web para prover recursos e suportar arquiteturas RESTful, dessa forma tem uma capacidade melhor de integração de EHRs [28].

FHIR é uma arquitetura modular e com recursos definidos, ou seja, toda troca de conteúdo é definida por um recurso, tem seu formato de dado específico e é possível de compreender os dados. Dessa forma, o FHIR permite a comunicação entre dispositivos e domínios clínicos de tecnologia, quebrando assim uma barreira na interoperabilidade [29]. Como essa arquitetura consegue se comunicar em JSON e XML, na Figura 2.1 é ilustrado um exemplo de formato de dados usado na comunicação para o recurso de paciente, os campos do exemplo são:

- *resourceType*: indica o tipo de recurso, ou seja, todo o objeto da figura é do mesmo tipo de recurso, que nesse caso é *Patient*;
- *id*: é o identificador único do recurso, ou seja, cada paciente tem um identificador que o sistema pode acessar ou criar para cada paciente;
- *active*: determina se aquele paciente está com cadastro ativo no sistema;
- *name*: aceita o cadastro de múltiplos nomes para o paciente, podendo realizar uma busca pelos diferentes nomes, e.g., nome de solteira e casada;
- *telecom*: são os registros de números de telefones para realizar contato caso preciso, descrevendo qual o uso do número e qual a ordem de prioridade de contato;
- *gender*: este campo aceita os valores de *male*, *female*, *other* e *unknown*;
- *_gender*: no caso do gender ter valor *other*, este campo é necessário para determinar qual é o gênero, nesse exemplo "x"indica não-binário;
- *birthDate*: é a data de nascimento no formato apresentado de ano/mês/dia;

[1] https://www.hl7.org/implement/standards/product_brief.cfm?product_id=185
[2] https://www.hl7.org/implement/standards/product_brief.cfm?product_id=186
[3] https://www.hl7.org/implement/standards/product_brief.cfm?product_id=7

Figura 2.1: Exemplo de JSON FHIR do recurso *Patient*



```
{
    "resourceType": "Patient",
    "id": "example",
    "active": true,
    "name": [
      {
        "use": "official",
        "family": "Chalmers",
        "given": [
          "Peter",
          "James"
        ]
      }
    ],
    "telecom": [
      {
        "use": "home"
      },
      {
        "system": "phone",
        "value": "(03) 5555 6473",
        "use": "work",
        "rank": 1
      }
    ],
    "gender": "other",
    "_gender": {
      "extension": [
        {
          "url": "http://example.org/Profile/administrative-status",
          "valueCodeableConcept": {
            "coding": [
              {
                "system": "http://terminology.hl7.org/CodeSystem/v2-0001",
                "code": "X",
                "display": "Ambiguous"
              }
            ]
          }
        }
      ]
    },
    "birthDate": "1974-12-25",
    "address": [
      {
        "use": "home",
        "type": "both",
        "text": "534 Erewhon St PeasantVille, Rainbow, Vic  3999",
        "line": [
          "534 Erewhon St"
        ],
        "city": "PleasantVille",
        "district": "Rainbow",
        "state": "Vic",
        "postalCode": "3999",
        "period": {
          "start": "1974-12-25"
        }
      }
    ]
}
```

Fonte: Adaptada do exemplo de recurso *Patient* [30].

- *address*: é o campo que indica os endereços do paciente;

O FHIR é dividido em recursos, na mensagem o recurso é definido pelo termo *resourceType* como apresentado na Figura 2.1. Alguns dos principais recursos do FHIR são:

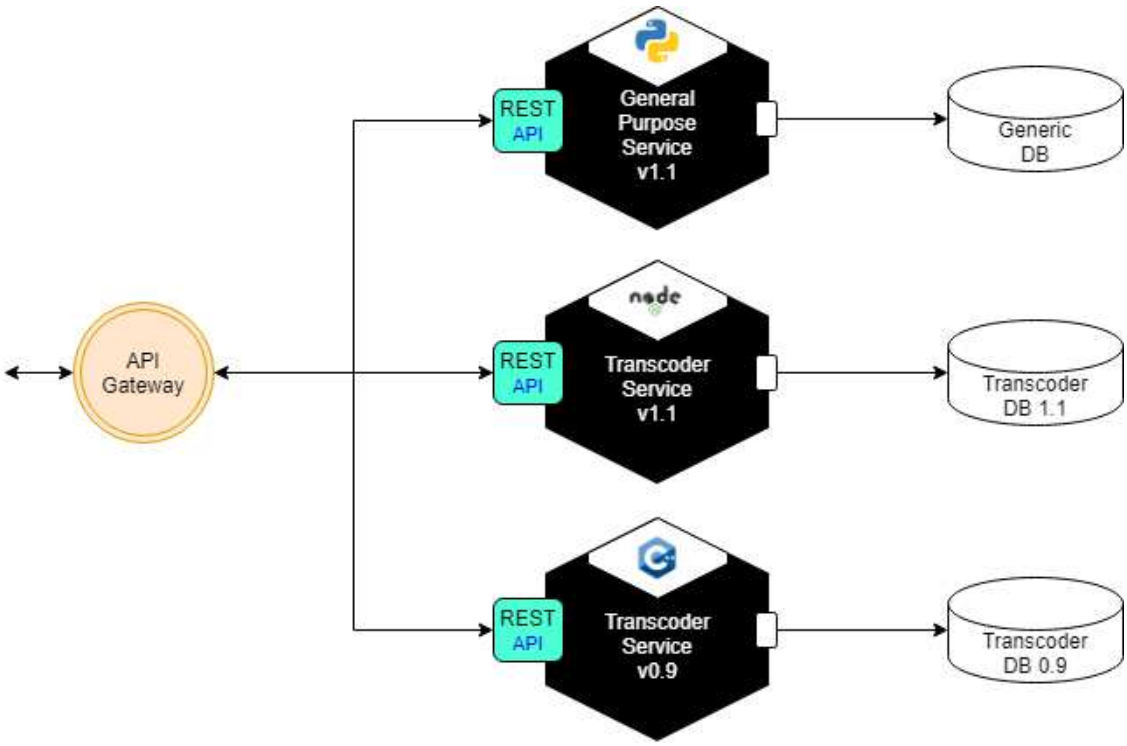
- *Patient*: define as informações do paciente;
- *Practitioner*: define todos os indivíduos que fazem parte dos sistemas de saúde, por exemplo, médicos, farmacêuticos, dentistas, enfermeiros, etc.;
- *Immunization*: registra o histórico de vacinas aplicadas no paciente com a data de aplicação;
- *Observation*: define o relatório das condições dos pacientes que estiveram em observação;
- *Specimen*: define uma amostra para estudo, por exemplo, uma amostra de bactéria coletada.

2.5 Microsserviços

Microsserviço é uma abordagem arquitetural que consiste em desmembrar uma arquitetura monolítica em várias partes menos, sendo assim, eles são autônomos, se comunicam entre si com protocolos leves e, geralmente, são desacoplados, ou seja, não dependem uns dos outros, mesmo que um deles não funcione, não necessariamente os outros também não vão estar funcionando. Dessa forma, os microsserviços podem ser implementados com linguagens diferentes, usar tecnologias diferentes, ter mais de uma versão do mesmo rodando no servidor, então o uso dessa arquitetura reduz o esforço de manutenção, aumenta disponibilidade, simplifica integração dos componentes, otimiza a escalabidade. Por isso, várias organizações migraram suas arquiteturas legadas para arquiteturas de microsserviços [11]. Um exemplo de arquitetura baseada em microsserviços é mostrado na Figura 2.2

Quando há uma grande utilização de serviços IoHT, ou seja, grande número de requisições na API, arquiteturas monolíticas apresentam problemas de tempo de resposta, devido

Figura 2.2: Exemplo básico de arquitetura de microsserviços



Fonte: Adaptada de [31]

ao escalonamento que precisa ser feito de cada serviço. Assim, um novo modelo de arquitetura, o de microsserviços, consegue realizar esse escalonamento, e também desacoplar serviços, permitir *containers* com diferentes linguagens para os microsserviços, tornando assim a arquitetura modular e de melhor manutenção [10].

2.5.1 Contêineres

Contêiner é uma tecnologia emergente de virtualização devido o seu desempenho ser maior que as máquinas virtuais [32]. Contêineres permitem que aplicações sejam executadas independentemente [32].

Um dos tipos de contêiner mais utilizados é o *Docker* [33]. Com *Docker*, não existe a necessidade de se preocupar com a execução da aplicação por alguma incompatibilidade com o sistema operacional do computador, pois o contêiner Docker isola a aplicação e executa toda a camada do sistema necessário para a mesma [33].

Outro contêiner é o *LXC*, que comparado ao Docker não contém a camada de *Docker*

engine, devido a isso, esse contêiner somente pode ser executado em sistemas Linux, e as aplicações dos contêineres devem também ser para o Sistema Operacional Linux [34].

2.6 Malha de Serviço

Malha de Serviço é uma camada da infraestrutura onde são executados os microsserviços, ou seja, nela é possível realizar configurações das interconexões dos serviços disponíveis. Nessa camada é definido o fluxo do tráfego de conexão, comunicação e observabilidade dos serviços. Dessa forma, é possível controlar acesso de usuários, determinar se serviços estão sendo executados, se alguns serviços estão sobrecarregados e então diminuir essa sobrecarga criando cópias do serviço, etc. [35].

A Malha de Serviço é uma ferramenta muito útil no contexto de microsserviços, por isso essa ferramenta foi escolhida para compor a solução proposta da implementação da arquitetura.

2.6.1 Istio

Istio[4] é uma Malha de Serviço que ajuda os desenvolvedores a configurarem uma infraestrutura de microsserviços. O Istio foi criado em 2018 em uma colaboração da Google, IBM e Lyft, provendo como algumas funcionalidades principais [36]:

- gestão de tráfego: fornece o roteamento de tráfego, tanto em um único cluster, quanto em múltiplos, via chamadas de API entre serviços;
- observabilidade: microsserviços gera uma grande complexidade de análise de telemetria para os serviços, devido sua grande quantidade de serviços. Dessa forma, o Istio tem algumas ferramentas de telemetria para prover ao usuário a possibilidade de trabalhar nelas;
- capacidades de segurança: O Istio cria um canal seguro de comunicação, dessa forma são fornecidas funcionalidades como criptografia TLS, autenticação, autorização e auditoria (AAA), outras também podem ser inseridas na arquitetura;

[4] https://istio.io/

- balanceador de carga: o Istio fornece um balanceador de cargas automático para as aplicações, realizando a distribuição da carga entre as cópias de cada microsserviço;
- camada de acesso plugável: esta camada de acesso permite controlar quais computadores podem acessar a aplicação ou não, essa camada não é obrigatória, podendo ser inserida/removida caso necessário;

2.7 Revisão Bibliográfica

Nesta seção é apresentada uma revisão bibliográfica sobre servidores de padrões de interoperabilidade e soluções de transcodificação, com objetivo de compreender as soluções existentes na literatura relacionadas ao tema desse trabalho.

Shoumik *et al.* [10] desenvolveram um servidor FHIR baseado em arquitetura de microsserviços, a desvantagem dessa solução em relação a com transcodificadores é a necessidade de mudar os sistemas para poder alcançar a interoperabilidade, dificultando a implantação dessa solução é que devido ao custo envolvido na implementação e mudança de sistema, não é atraente para os centros médicos que já têm seus sistemas. Segundo os autores, as vantagens desse servidor de microsserviços em relação aos servidores monolíticos são:

- escalabilidade: capacidade de escalar horizontalmente os serviços específicos baseados na carga dos mesmos;
- *service deployment*: realizar a implementação de serviços e versões dos mesmos de forma mais prática, enquanto os outros serviços continuam com seu funcionamento normal, nesse ponto há a possibilidade de executar mais de uma versão do mesmo serviço, enquanto são realizadas mudanças de versões e testes das mesmas;
- desacoplamento: diminui o acoplamento entre serviços, ou seja, cada serviço é executado independente do outro, mesmo que um pare de ser executado, os outros ainda continuam em funcionamento;
- agilidade no desenvolvimento: devido aos microsserviços terem acessos baseados em REST, cada um tem em sua estrutura o necessário para funcionar sem depender dos outros, sendo assim é possível utilizar linguagens de programação diferentes, times

diferentes, desde que os critérios da API sejam atendidos, o servidor irá funcionar sem problemas;

- alta disponibilidade e API: como os microsserviços conseguem escalar, então a disponibilidade de cada serviço aumenta, pois além de não depender dos outros serviços, existe mais de um serviço do mesmo tipo executando, além disso, têm as vantagens de utilizar API, descrita na seção 2.3.

Mesmo com essas vantagens, microsserviços tem a desvantagem de a criação dos mecanismos de comunicação entre serviços serem complexos [37], que pode ser solucionado usando malha de serviço. O problema dessa solução é que mesmo que sejam criados servidores FHIR, os servidores que já existem com seus padrões, os ditos legados, não conseguem se comunicar diretamente. Com isso, ainda existe a necessidade de alterar os sistemas legados para o mesmo padrão, no caso o FHIR, então a solução com transcodificadores que já conecta os sistemas suportados.

Naveed *et al.* [16] criaram um *framework* para prover interoperabilidade semântica, ou seja, ao enviar dados de um sistema para outro, manter o mesmo significado no receptor, sem a necessidade de intervenção humana. O problema dessa solução automatizada por inteligência artificial em relação a um transcodificador manualmente construído é que na primeira solução não há como saber com exatidão se o dado foi interpretado corretamente ou não, somente em testes, já no transcodificador isso já é definido.

O *NextGen Connect Integration Engine* é uma solução aberta que ajuda entidades a alcançarem interoperabilidade [38]. Nele é possível fazer a integração de diferentes formatos como XML[5] , HL7 v 2.x e 3.X –CCD/CCDA FHIR , DICOM[6], NCPDP[7], X12[8], entre outros [38]. Entretanto, essa solução não é em microsserviços, perdendo algumas vantagens de, por exemplo, somente inserir o microsserviço na malha de serviço para ser utilizado, e também não tem funcionalidades como sincronizar dados, ou seja, solicitar, a várias entidades conectadas e registradas em um barramento, o envio de dados para atualizar os dados dos pacientes no barramento de saúde.

[5] https://www.w3.org/XML/
[6] https://www.dicomstandard.org/
[7] https://www.ncpdp.org/
[8] https://x12.org/flow/health-care

Li *et al.* [39] desenvolveram uma *interface engine* que integra IEEE 11073 DIM com FHIR utilizando o protocolo CoAP. Essa solução resolve parcialmente o problema entre dois servidores que utilizem esses formatos de dados, pois a escalabilidade do sistema não foi descrita e a escalabilidade desse sistema não é por microsserviços. Nesse caso, para uma grande quantidade de requisições e para uma mesma quantidade de recursos computacionais, uma arquitetura em microsserviços potencialmente apresenta um tempo de resposta melhor. Nesse trabalho também não foi descrito como seria implementado para outros protocolos e se é possível.

Andersen *et al.* [29] implementaram uma arquitetura para mapear *Integrated Crinical Environment* (ICE) e FHIR. Esse trabalho mostra a viabilidade de mapear IEEE 11073 que é um exemplo de arquitetura ICE, mas esse mapeamento não tem as vantagens de utilizar microsserviços para flexibilidade do sistema.

Mukhiya *et al.* [40] apresentaram uma arquitetura baseada em nuvem para prover interoperabilidade, e os pacientes poderiam interagir diretamente com os registros médicos. Nesse trabalho, é realizada a implementação utilizando apenas o FHIR, essa tentativa de definir um padrão central é um ponto de falha devido à resistência de todos os provedores utilizarem o mesmo padrão, além da implementação ser monolítica.

Kumar *et al.* [41] propuseram o estado da arte na segurança de dados na troca de mensagens em sistemas interoperáveis, nesse caso os sistemas utilizam *blockchain*. No caso dos microsserviços contidos na malha de serviço, esses são isolados e protegidos de acesso externo via autenticação, autorização, criptografia TLS, além de robusto, torna a implementação do sistema mais simples e ágil.

Ayaz *et al.* [42] realizaram uma revisão sistemática sobre o padrão de saúde FHIR. Essa revisão contém implementações e desafios de 80 trabalhos, com vários cenários de aplicação. Mas esses sistemas não conseguem prover interoperabilidade com os outros padrões. Pois, não existe compatibilidade direta entre FHIR e os outros padrões de saúde, por exemplo, HL7v2. Dessa forma, ainda se faz necessário o uso de transcodificadores entre esses sistemas de saúde para prover interoperabilidade. Isso se deve ao fato de que para o FHIR prover interoperabilidade todos os sistemas deveriam utilizar o mesmo padrão de saúde, acarretando um custo associado para mudança dos sistemas, na prática.

Esta revisão demonstrou que não somente buscar interoperabilidade é essencial, mas

também provê-la de forma simplificada e eficiente, ou seja, através do uso de transcodificadores entre padrões de saúde, comunicação com baixo tempo de resposta, disponibilidade dos recursos e flexibilidade de escolha dos padrões.

Como os requisitos da aplicação e os transcodificadores de padrões de saúde são importantes, uni-los em uma única especificação é fundamental. Devido a isso, no Capítulo 3 é apresentada uma proposta de solução que considera esses dois critérios.

Capítulo 3

Modelo Arquitetural

Neste capítulo, apresenta-se a arquitetura da plataforma de transcodificação para interoperabilidade de dados de saúde. Na Seção 3.1 são discutidas as premissas para a solução arquitetural. Na Seção 3.2 apresenta-se a arquitetura geral do sistema. Na Seção 3.3 é detalhado o funcionamento dos microsserviços e seus módulos. Na seção 3.4 é explicada a regra de transcodificação para simplificar a implementação. Na Seção 3.5 é apresentado o fluxograma detalhado da arquitetura. Na Seção 3.6 é descrito os pontos principais do capítulo.

3.1 Premissas Arquiteturais

A solução proposta consiste em criar um modelo arquitetural de transcodificação entre formatos de dados de saúde. Para tanto foram mapeadas as seguintes premissas para a definição da arquitetura:

- (a)*transcodificar dados*: transformar uma estrutura de dados de um padrão de saúde em outro, para viabilizar a comunicação entre sistemas com padrões de saúde diferentes;
- (b)*escalabilidade*: a arquitetura deve conter uma forma simples de escalar a aplicação, ou seja, a depender da carga a qual está submetida o transcodificador, sem escalar partes desnecessárias;

- (c)*flexibilidade de escolha de padrões*: a arquitetura deve permitir adicionar ou remover diferentes padrões e versões de forma simplificada, por exemplo, executando um único comando, de modo que os outros serviços continuem suas execuções sem nenhuma interrupção;

- (d)*flexibilidade de inserir novos provedores de saúde*: deve ser possível inserir provedores de saúde tanto solicitantes, quanto destinatários da transcodificação.

- (e)*disponibilidade dos serviços*: devido à existência de dados de monitoramento que são de quadros clínicos graves, surge a necessidade do sistema estar disponível para atender essa demanda quando necessário.

Tabela 3.1: QoS para dados de saúde de dispositivos médicos

Tipos de dados	Confiabilidade	Latência do serviço
Alarmes (tempo real)	++++	<3 s
Busca por prontuários médicos	++	<5 s
Formas de ondas(tempo real)	+++	<3 s
Acesso via Web	++	<5 s

confiabilidade: média (++), alta (+++), altíssima/essencial (++++)

Fonte: Adaptado de [9]

A Tabela 3.1 contém as informações de qualidade de serviço do padrão IEEE 11073 [9], essas referenciadas na Tabela são análogas aos outros padrões, ou seja, esses requisitos devem ser satisfeitos pelas implementações. Analogamente, esses valores podem ser utilizados como base para demonstrar a importância de seguir os requisitos, e avaliação de desempenho.

Existem milhares de requisições por segundo de saúde sendo enviados e recebidos, e em alguns casos, como pandemias, isso pode aumentar significativamente [43]. Para conseguir lidar com isso no lado do servidor, Betoni *et al.* [13] elaboraram uma plataforma FHIR, e devido à característica da qualidade de serviço (QoS), eles optaram por uma arquitetura de microsserviços, a qual eles comprovaram que supre a demanda de serviços de saúde utilizando FHIR. Como a transcodificação é uma camada entre o servidor e cliente, essa camada não pode inserir uma grande latência na comunicação. Dessa forma, para a premissa (b) também se faz necessário o uso de microsserviços nos transcodificadores, pois, essa vasta

quantidade de requisições que trafega entre servidores e clientes não pode ser impactada por atrasos gerados por uma camada adicional, evitando impacto na qualidade de serviço, por exemplo, os que são apresentados na Tabela 3.1.

Com o passar do tempo, padrões de saúde são criados e também atualizados [42], com isso, surge a necessidade de atualizar os transcodificadores, mas também de ter disponível versões anteriores dos mesmos. Além disso, é importante que ao adicionar esses novos serviços, não impacte no funcionamento do sistema. Para isso, para suprir a premissa (c), microsserviços também podem ser utilizados, pois, a adição e remoção de versões e tipos de transcodificadores é realizada sem impacto nos demais serviços, devido ao desacoplamento.

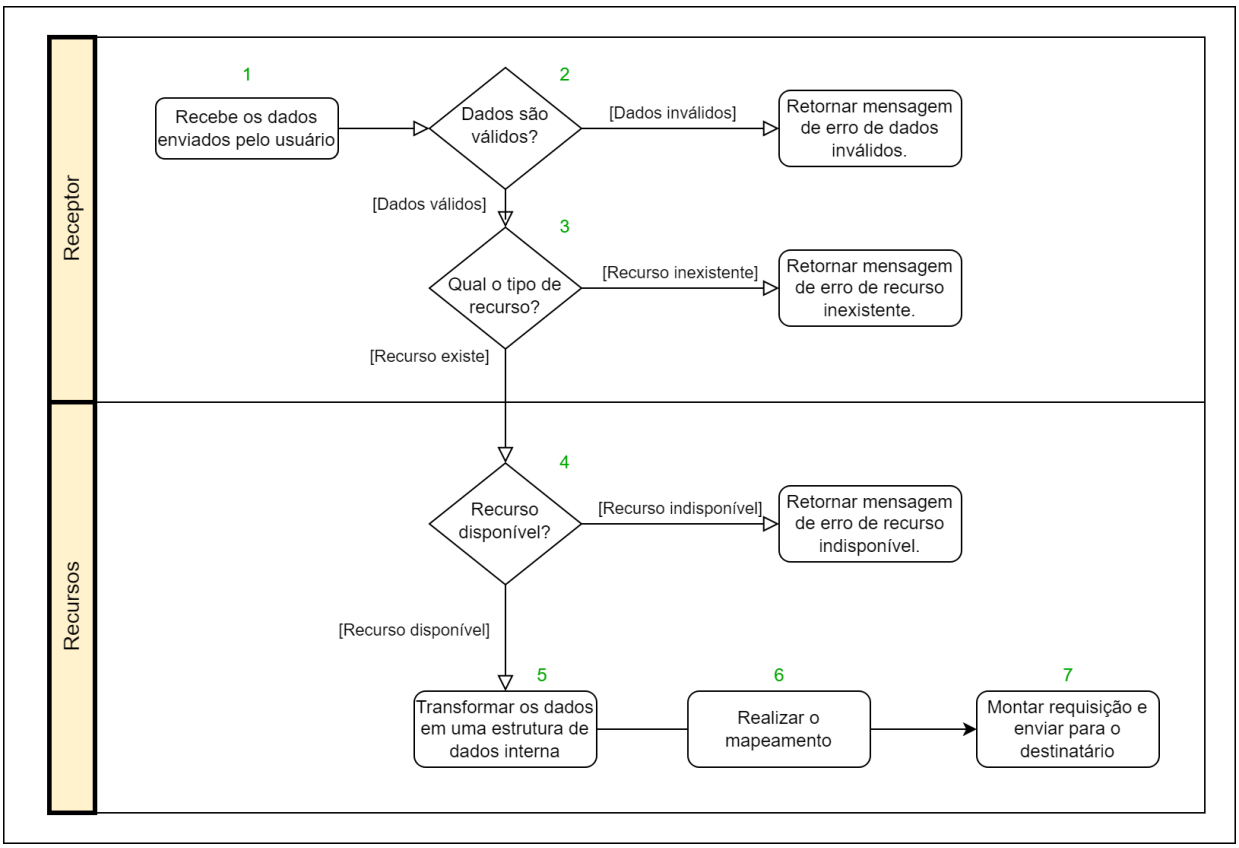
Para a premissa (d), advém a necessidade de ampliar ou reduzir o acesso externo aos transcodificadores, ou seja, que provedores de saúde possam ter ou não acesso aos transcodificadores. Devido a isso, o uso de uma malha de serviço de microsserviços é recomendado, pois é possível determinar os provedores que podem ter ou não acesso com um canal de comunicação seguro em relação aos monolíticos [43].

Sistemas de saúde precisam garantir disponibilidade, ou seja, os serviços devem funcionar a maior parte do tempo de modo intermitente, de acordo com seus requisitos de disponibilidade de serviço [9]. Desse modo, microsserviços podem aumentar a disponibilidade do sistema de saúde, como apresentado por Liu *et al.* em [12]. Devido ao fato dos transcodificadores fazerem parte de uma camada adicional na comunicação, é importante que os mesmos tenham disponibilidade, pois se o serviço estiver indisponível, não cumprirá os requisitos de qualidade de serviço. Portanto, para a cumprir a premissa (e) de disponibilidade, o sistema de transcodificadores deve ser implementado em microsserviços. Nesta premissa (e), também se faz necessário o uso da malha de serviço, a qual oferece observabilidade ao sistema, com isso é possível saber se os microsserviços estão cumprindo o seu papel, e também de controlar a plataforma para alcançar as métricas necessárias, como apresentando por Nunes *et al.* [44].

A premissa (a) é a funcionalidade base do sistema, a qual pode ser alcançada utilizando implementações monolíticas ou microsserviços. Mas devido às vantagens da utilização de microsserviços e das outras premissas (b-e), a arquitetura base deste trabalho fica definida sendo microsserviços inserida dentro de uma malha de serviço.

3.2 Arquitetura

Figura 3.1: Diagrama do Fluxo da arquitetura.



Fonte: Elaborada pelo autor.

Apresenta-se na Figura 3.1 o fluxograma de execução da transcodificação do sistema realizado pela arquitetura proposta. No fluxograma são realizadas até sete etapas-base, as quais são descritas a seguir.

Na etapa 1 os dados enviados pelo usuário são recebidos pelo serviço receptor para serem processadas.

Na etapa 2 é verificado se os dados recebidos são válidos. A validade dos dados é definida pelo padrão de saúde, então deve-se seguir o formato correto e ter enviado todas as informações obrigatórias para a transação. Por exemplo, a requisição FHIR apresentada na Figura 2.1, caso não fosse enviado com o campo "*resourceType*", o servidor que recebesse essa requisição não teria a informação de qual é o tipo de recurso que está sendo trabalhado, logo não seria possível realizar a requisição.

Na etapa 3 após a validação dos dados, obtém-se qual é o tipo de recurso, por exemplo, é apresentado na Figura 2.1 o recurso paciente. Com isso, é possível verificar se existe a implementação deste recurso internamente, pois o sistema pode não estar completo ou uma atualização do padrão de saúde, o FHIR neste exemplo, possa ter adicionado um novo recurso, que ainda não foi implementado naquela versão do transcodificador. Em caso de não haver a implementação do recurso, é retornada uma mensagem de erro indicando que naquela versão ainda não existe o recurso.

Na etapa 4 é realizada uma verificação de disponibilidade do serviço do recurso ao qual está sendo solicitada a transcodificação, por exemplo, recurso paciente do FHIR para paciente do *HL7 v2*. Caso o recurso não esteja disponível, é retornada uma mensagem de erro indicando essa indisponibilidade.

Na etapa 5 é realizada a transformação dos dados recebidos em uma estrutura de dados interna para ser possível realizar a manipulação dos dados, por exemplo, transformar a mensagem de texto conforme apresentada na Figura 3.2 recebida em uma estrutura de dicionário mostrada na Figura 3.3.

Figura 3.2: Exemplo de cabeçalho HL7v2.

```
MSH|^~\&|FROM_APP|FROM_FACILITY|TO_APP|TO_FACILITY|20180101000000||ADT^A01|20180101000000|P|2.5|
```

Fonte: Adaptada de [45].

Na etapa 6 utiliza-se a estrutura dados obtidos na etapa 5 para transformar no formato do padrão de saúde alvo a ser enviado para o destino, ou seja, é realizada a leitura de cada campo e insere-se a informação no padrão alvo. Por exemplo, ao ler a informação da Figura 3.3, é possível obter qual o provedor que enviou a mensagem (“FROM_FACILITY”) na linha 7 da mesma Figura.

Na etapa 7 os dados obtidos na etapa anterior são incrementados com informações necessárias a depender do padrão de saúde o qual deseja-se transformar, por exemplo, criar um identificador único universal (UUID) para ser utilizado como identificador para cada novo objeto de recurso FHIR, conforme apresentado no campo “*id*” da Figura 3.9.

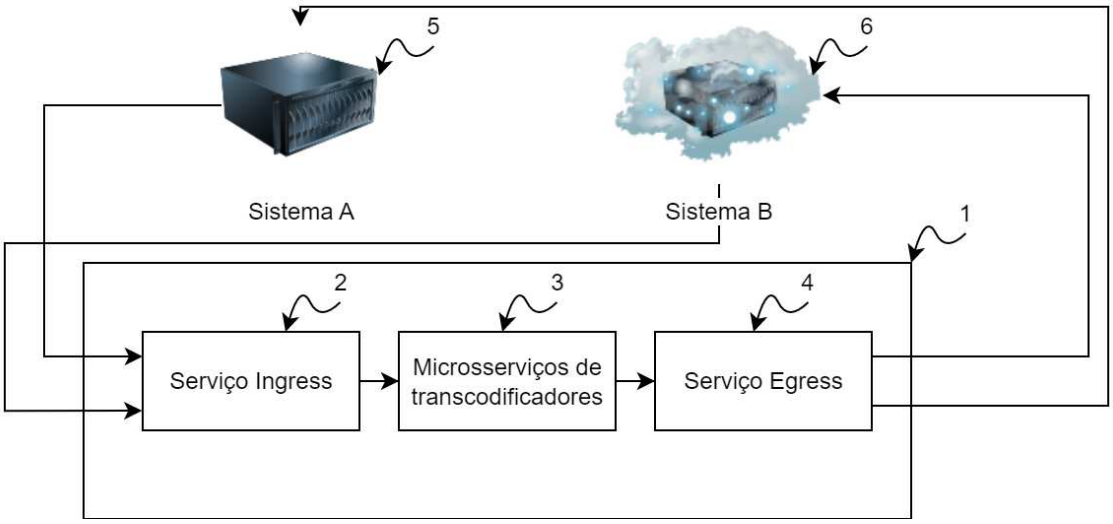
Apresenta-se na Figura 3.4 um diagrama simplificado da arquitetura de componentes da arquitetura proposta. Essa arquitetura geral é baseada na *interface engine* criada pela

Figura 3.3: Exemplo de estrutura JSON contendo os dados recebidos.


```
"segments": [
  {
    "segmentId": "MSH",
    "fields": {
      "5": "TO_FACILITY",
      "2": "FROM_APP",
      "3": "FROM_FACILITY",
      "0": "MSH",
      "1": "^~\\&",
      "10": "P",
      "4": "TO_APP",
      "9": "20180101000000",
      "8.1": "ADT",
      "11": "2.5",
      "8.2": "A01",
      "6": "20180101000000"
    }
  },
```


Fonte: Adaptada de [45].

Figura 3.4: Diagrama simplificado do modelo arquitetural



Fonte: Elaborada pelo autor.

NextGen healthcare [38]. Nessa arquitetura de microsserviços, destacam-se os seguintes componentes:

- *Servidor de borda de multi-acesso ou servidor de computação em nuvem* (1): centraliza os serviços onde a aplicação é executada;
- *Serviço Ingress* (2): serviço responsável por receber requisições externas ao item 1 e direcioná-las para o seu devido microsserviço, ou rejeitá-la caso o usuário não tenha permissão de acesso. Esse serviço se baseia na carga de utilização dos microsserviços para decidir para quem enviar a requisição;
- *Microsserviços de transcodificadores* (3): microsserviços que transformam os dados para o tipo de dados alvo, ou seja, para onde será enviado;
- *Serviço Egress* (4): serviço que permite os serviços internos se comunicarem com serviços externos;
- *Sistema A* (5): é um sistema que utiliza um padrão de saúde definido, por exemplo, HL7 v2;
- *Sistema B* (6): este outro sistema, utiliza um padrão de saúde diferente do Sistema A, que pode ser o FHIR, por exemplo.

O seguinte exemplo serve para exemplificar o funcionamento desses módulos: um agente utilizando o sistema HL7 v2 deseja cadastrar um paciente em um servidor FHIR, mas devido à incompatibilidade direta fica viável a utilização transcodificador apresentado na Figura 3.4; o agente do *Sistema A* (5) então envia a mensagem HL7 v2 de cadastro para o *Serviço Ingress* (2), o qual envia a mensagem para os *Microsserviços de transcodificadores* (3), após a transcodificação essa mensagem é enviada para o *Sistema B*, o qual executa a operação de cadastrar o paciente em seu sistema.

Nas próximas seções serão descritos os principais componentes da arquitetura apresentada nesta seção.

3.3 Microsserviços de Transcodificação

Esta seção descreve o módulo de *Microsserviços de transcodificadores* (módulo 3 da Figura 3.4), a enumeração desta seção é referente aos itens apresentados no diagrama da Figura 3.5. Os *receptores* (30) têm o objetivo de receber requisições externas, validar os dados e enviar para seu(s) devido(s) microsserviço(s) de transcodificação de dados. O receptor *A_to_B* (31), tem o papel de receber a mensagem do padrão utilizado pelo sistema A, validar a mensagem baseada na especificação do padrão de saúde e enviar para o(s) devido(s) microsserviço(s) de recurso(s) contido em *Recursos A_B* (33), por exemplo, pode representar um receptor de HL7v2 para FHIR.

Módulos *Recursos A_B* (33) englobam os microsserviços referentes a transcodificação de recursos do padrão A para os recursos equivalentes do recurso B. O módulo *ResA0_ResB0* (34) transcodifica o primeiro recurso A0 no primeiro recurso B0, por exemplo, é o transcodificador do recurso *PID* do HL7v2, como apresentado na Lista 2.1, para o recurso *Patient* do FHIR, como ilustrado na Figura 2.1. O módulo *ResAn_ResBn* (35) representa o n-ésimo e último recurso a ser transcodificado do padrão de saúde A para o padrão B.

O receptor *B_to_A* (32) é o mapeamento inverso do módulo *A_to_B* (31). O receptor (32) recebe a mensagem no padrão utilizado pelo sistema B, valida a mensagem recebida, e envia para o(s) recurso(s) correto(s) em *Recursos B_A* (36).

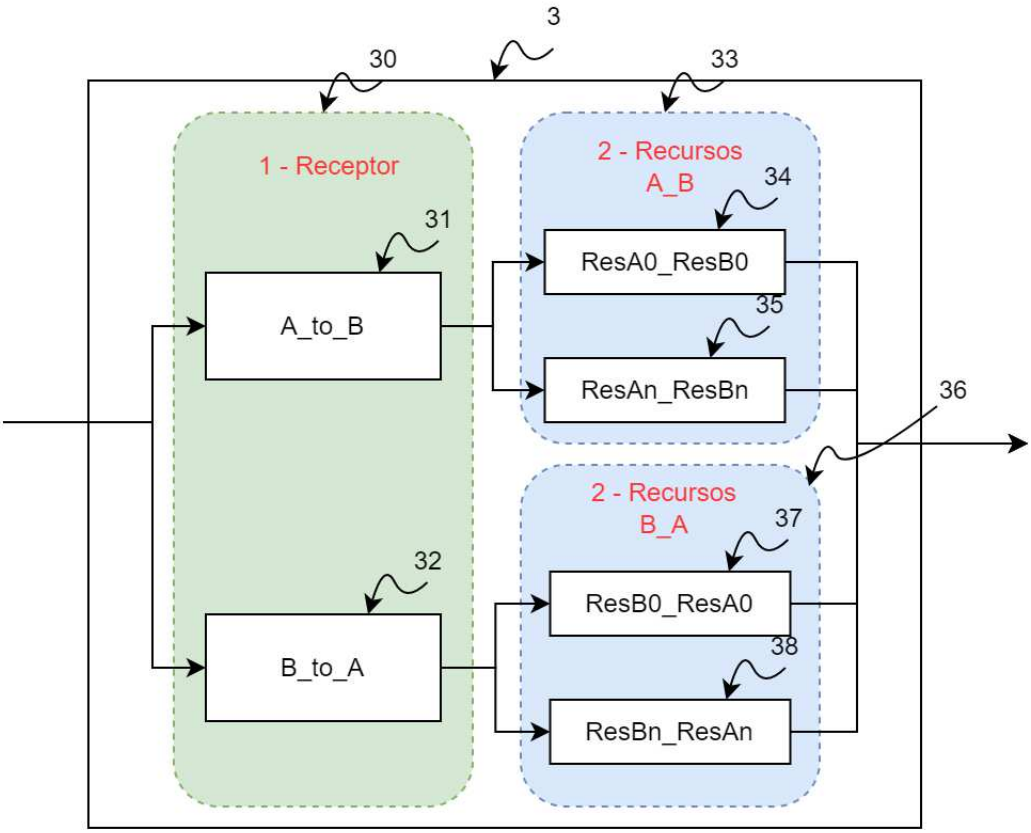
Os módulos *Recursos B_A* (36) contêm os mapeamentos inversos dos módulos presentem no *Recursos A_B* (33), criando assim, um transcodificador bidirecional entre os dois sistemas.

3.3.1 Microsserviço Receptor

O diagrama apresentado na Figura 3.6 representa um módulo contido no *Receptor* (30) da Figura 3.5, ou seja, é o diagrama da arquitetura dos receptores, os quais são descritos da seguinte forma:

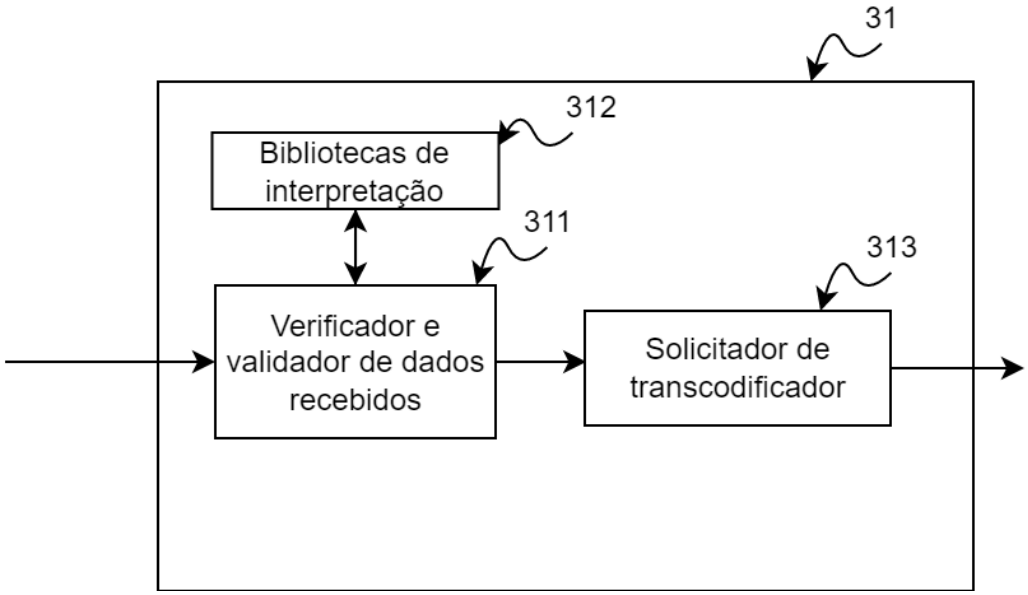
- *Verificador e validador de dados recebidos* (311): esse módulo, utilizando as *bibliotecas de interpretação* (312), faz a verificação se os dados estão no formato correto e têm todos os campos obrigatórios, assegurando que seja possível trabalhar na transcodifi-

Figura 3.5: Diagrama dos microsserviços de transcodificação.



Fonte: Elaborada pelo autor.

Figura 3.6: Diagrama do microsserviço receptor



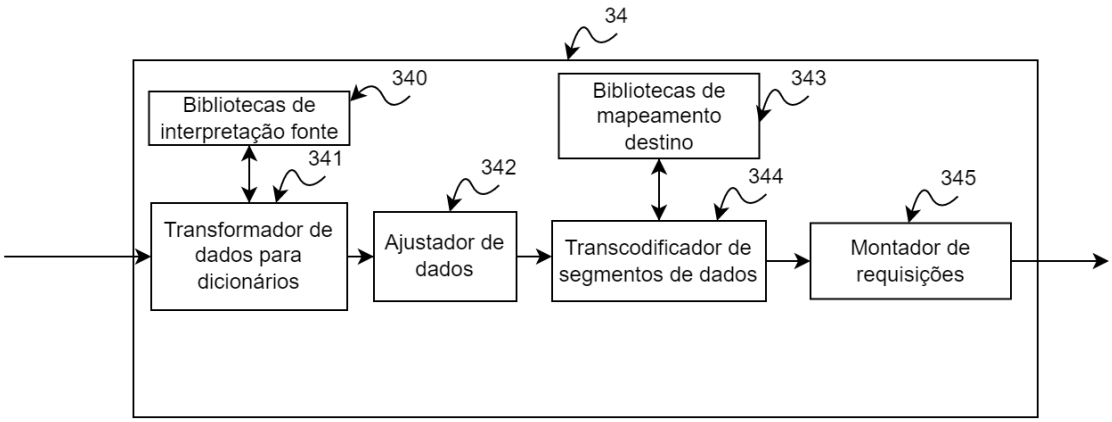
Fonte: Elaborada pelo autor.

cação;

- *Bibliotecas de interpretação* (312): essas bibliotecas são referentes aos dados do padrão de saúde que foram enviados, sendo possível trabalhá-los na linguagem de implementação, verificando a sua validade e realizando a leitura dos campos corretamente;
- *Solicitador de transcodificador* (313): identifica o módulo de transcodificação e faz a solicitação dos transcodificadores corretos no *Recursos A_B*, (33) ou no *Recursos B_A* (36) da Figura 3.5.

3.3.2 Microsserviço Transcodificação de Recurso

Figura 3.7: Diagrama do microsserviço de transcodificação de recurso.



Fonte: Elaborada pelo autor.

No diagrama da Figura 3.7 é ilustrado um microsserviço de transcodificação de recurso, esse é o módulo 34 apresentado na Figura 3.5, e o mesmo é composto pelos seguintes componentes:

- *Bibliotecas de interpretação fonte* (340): interpreta os dados do formato da fonte (os dados que foram recebidos) em estruturas de dados da linguagem utilizada na implementação;
- *Transformador de dados para estrutura de dados interna* (341): este componente é responsável por transformar, utilizando as *Bibliotecas de interpretação fonte* (340),

os dados recebidos em uma estrutura de dados interna da linguagem utilizada para simplificar a manipulação das informações;

- *Ajustador de dados* (342): realiza os ajustes dos dados nos campos que forem necessários, a depender dos campos dos recursos, por exemplo, é apresentado na Figura 3.8 o identificador ('*identifier*') que deve ser tratado para poder ir para o formato FHIR apresentado na Figura 3.9, que indica uma *url* para o sistema do identificador utilizado;

- *Bibliotecas de mapeamento* (343): bibliotecas para auxiliar na transcodificação e reutilização em recursos semelhantes;

- *Transcodificação de segmentos de dados* (344): Esse módulo realiza a transcodificação dos dados contidos na estrutura de dados interna para o padrão alvo, por exemplo, transforma os dados da Figura 3.8 na estrutura da Figura 3.9, obtendo assim, o recurso no devido formato do padrão de saúde alvo;

- *Montador de requisições* (345): monta a requisição baseada nos segmentos de dados obtidos no módulo (344).

Figura 3.8: Exemplo de dicionário contendo dados de paciente.

```
person = {
    'name': 'Elise',
    'family_name': 'Wong',
    'birth_date': '1983-06-15',
    'identifier': [{
        'value': "90012",
        'system': "NIST-MPI-1",
        'type': "MR"
    }]
}
```

Fonte: Elaborada pelo autor.

Figura 3.9: Exemplo de JSON contendo parte do recurso de paciente que foi transcodificado.

Fonte: Elaborada pelo autor.

3.4 Regra de Transcodificação

A regra geral de transcodificação é descrita em duas partes:

- *Algoritmo Receptor*: o algoritmo do módulo *Receptor* (30) da Figura 3.5 é apresentado no Algoritmo 1;
- *Algoritmo microsserviço transcodificação de recurso*: este algoritmo do módulo *Recursos A_B* (33) da Figura 3.5 é apresentado no Algoritmo 2.

A primeira regra de transcodificação é apresentada no Algoritmo 1. O receptor faz a validação dos dados que foram recebidos pela plataforma, verifica qual é o tipo de recurso da mensagem e solicita ao microsserviço de transcodificação que a realize. O Algoritmo 1 é descrito, por linha, da seguinte forma:

- (1): recebe os dados da mensagem enviados pelo usuário e salva na variável *recurso*;
- (2): verifica se os dados da variável *recurso* são válidas em relação ao padrão de saúde utilizado, por exemplo, HL7 v2;

- (3): caso os dados sejam válidos em (2), armazena qual o tipo de recurso enviado pelo usuário e armazena na variável *tipo_de_recurso*;
- (4): verifica se o tipo de recurso está implementado na plataforma;
- (5): caso o tipo de recurso exista em (4), realiza a requisição para o *microsserviço de transcodificação* referente ao tipo de recurso enviado pelo usuário;
- (6): retorna a resposta do microsserviço para o usuário;
- (7): fluxo executado quando o recurso não está implementado na verificação (4);
- (8): retorna mensagem de erro indicando que o recurso não está implementado na plataforma;
- (9): fim da verificação (4);
- (10): fluxo executado quando os dados enviados pelo usuário são inválidos, ou seja, está fora do formato definido pelo padrão de saúde, ou está faltando campos obrigatórios;
- (11): retorna mensagem de erro que os dados são inválidos, indicando o erro que ocorreu;
- (12): fim da verificação (2).

Algoritmo 1 Algoritmo Receptor.

```
1: recurso = recebe_dados_do_recurso()
2: if dados_são_válidos(recurso) then
3:     tipo_de_recurso = qual_o_tipo_de_recurso(recurso)
4:     if tipo_de_recurso_está_implementado(tipo_de_recurso) then
5:         response = requisição_para_o_recurso(recurso)
6:         return response
7:     else
8:         return erro_de_recurso_não_implementado
9:     end if
10: else
11:     return erro_de_dados_inválidos
12: end if
```

A segunda regra de transcodificação é apresentada no Algoritmo 2. O transcodificador recebe os dados do recurso, transforma a estrutura de dados para manusear os dados, ajusta os dados, monta a requisição e envia para o destino.

O Algoritmo 2 é descrito, por linha, da seguinte forma:

- (1): recebe os dados advindo do *Receptor*;
- (2): transforma a estrutura de dados, recebida, na estrutura que facilite a leitura dos dados da mensagem recebida;
- (3): ajusta os dados colocando nas unidades corretas, e insere dados do padrão de saúde alvo que não existem na mensagem recebida;
- (4): mapeia os dados da estrutura para o padrão de saúde alvo, geralmente este padrão é o FHIR, pela possibilidade de utilizar o FHIR como padrão central;
- (5): realiza a requisição para o servidor destino, enviando a mensagem obtida em (4);
- (6): retorna a reposta do servidor da requisição (5).

Algoritmo 2 Algoritmo microsserviço transcodificação de recurso.

```
1: recurso = recebe_dados_do_recurso()
2: dados = transforma_estrutura_de_dados(recurso)
3: dados = ajustador_de_dados(dados)
4: mapeado = mapeia_dados_para_padrão_alvo(dados)
5: response = requisição_para_servidor_destino(mapeado)
6: return response
```

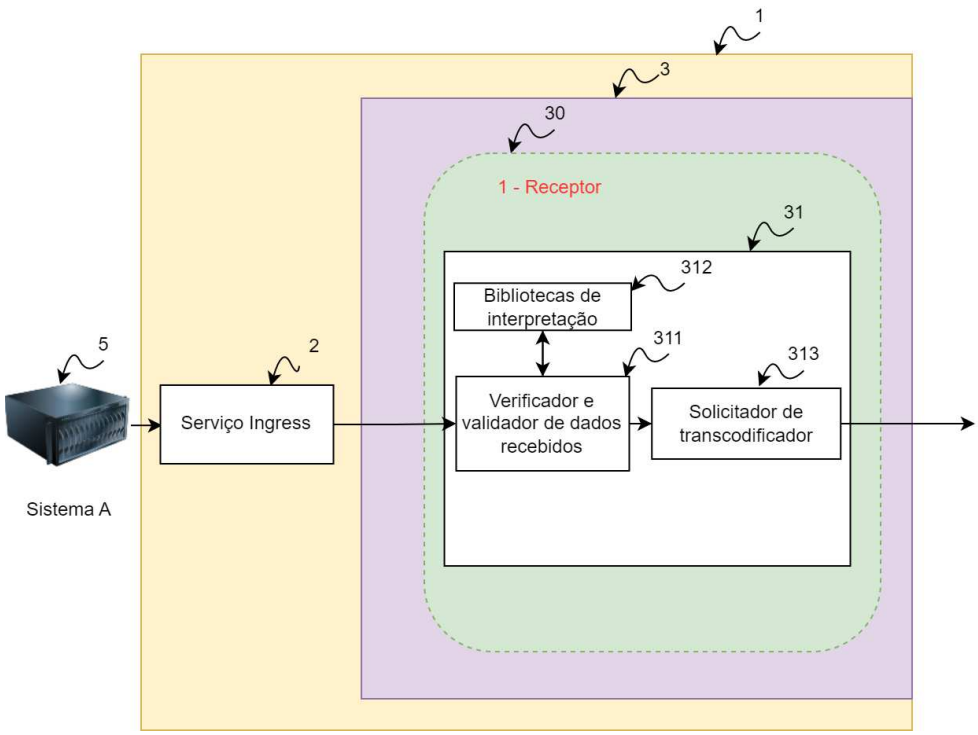
Uma regra simplificadora da transcodificação é utilizar um padrão central. Esse padrão central funciona como uma camada intermediária, por exemplo, um transcodificador HL7v2 para FHIR com um transcodificador FHIR para CCDA consegue transformar os dados de HL7v2 para CCDA, sem a necessidade de implementar transcodificadores HL7v2 para CCDA e CCDA para HL7v2. Dessa forma, fica definida a utilização do FHIR como padrão central, pois o mesmo tem uma tendência a crescimento e grande potencial, sendo um padrão completo [46].

Em resumo, a regra determina: somente deve-se implementar transcodificadores entre padrões de saúde e FHIR para simplificar a implementação, o custo de implementação e o

tempo de implementação. Com isso, caso um sistema HL7v2, por exemplo, tenha o objetivo de transcodificar em CCDA, no módulo (345) *Montador de requisições* da Figura 3.7 deve conter a informação de que o destino indicado não é FHIR. Com isso, esse módulo realiza requisição para o próprio servidor de transcodificação para realizar a transcodificação de FHIR para CCDA. Após obter a mensagem no padrão alvo, a mensagem é enviada para o sistema CCDA.

3.5 Fluxo Detalhado da Arquitetura

Figura 3.10: Primeira parte do diagrama simplificado do modelo arquitetural contendo todos os módulos.

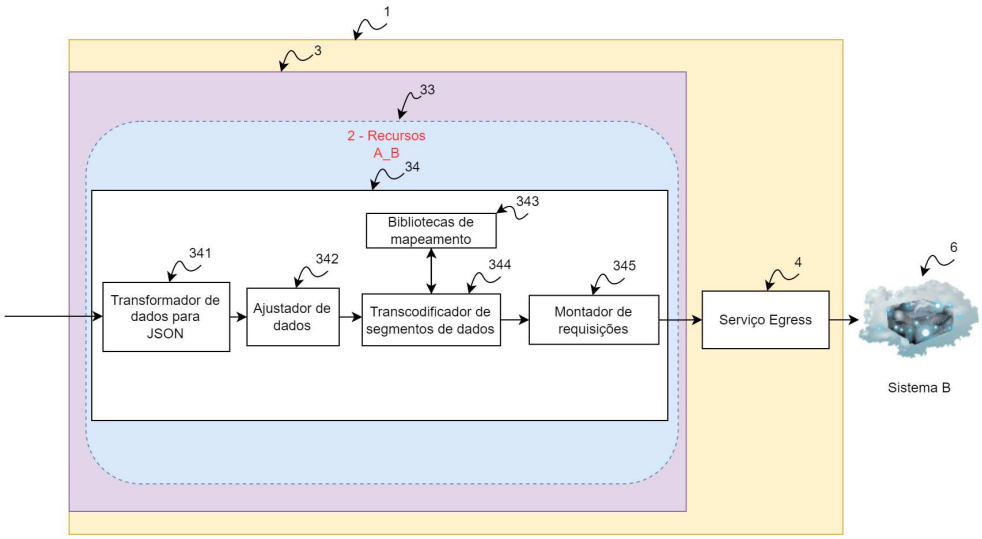


Fonte: Elaborada pelo autor.

Nesta seção é apresentado um fluxo de transcodificação, baseado nos módulos e enumeração apresentados nas Figuras 3.10 e 3.11, de uma mensagem enviada do *Sistema A* (5), que usa o padrão HL7v2, para o *Sistema B* (6), que utiliza o padrão FHIR.

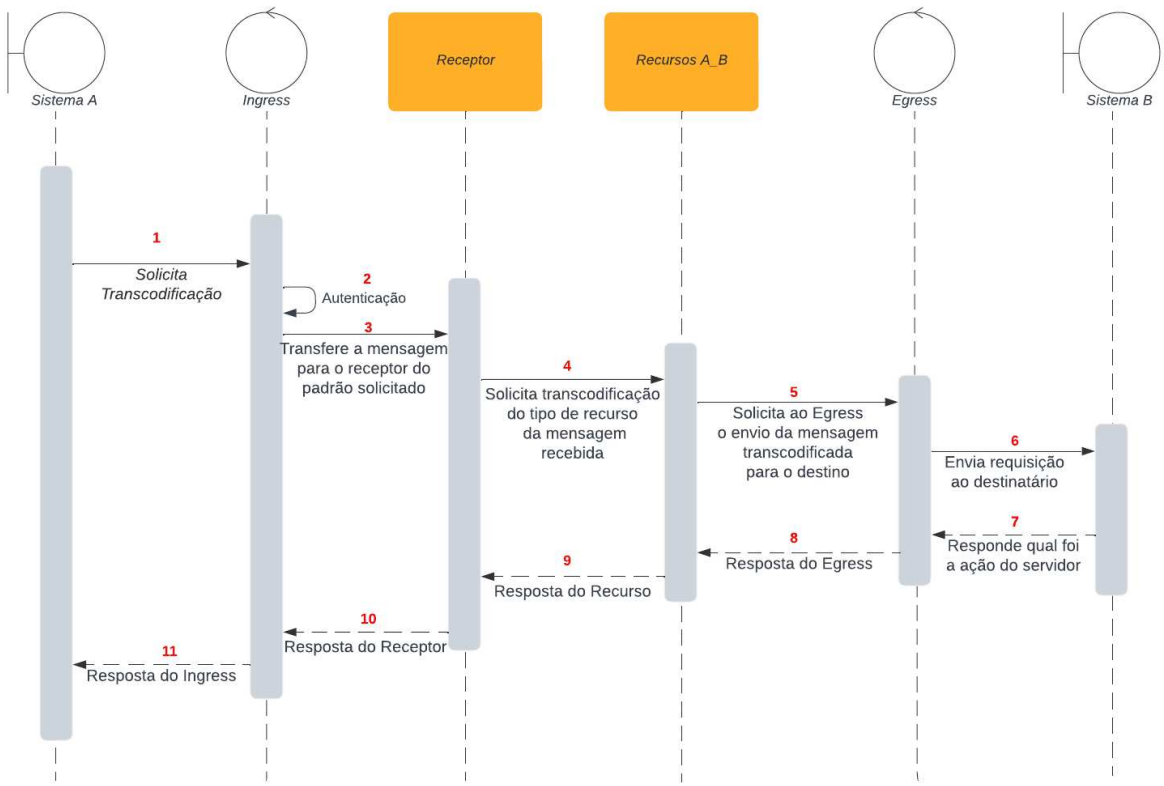
Para esse exemplo temos a seguinte configuração:

Figura 3.11: Segunda parte do diagrama simplificado do modelo arquitetural contendo todos os módulos.



Fonte: Elaborada pelo autor.

Figura 3.12: Diagrama de sequência da arquitetura.



Fonte: Elaborada pelo autor.

- *Servidor de borda de multi-acesso ou servidor de computação em nuvem*(1): é um servidor de borda; *transcoder_server.com* é o nome do domínio utilizado;
- *Sistema A* (5): utiliza o padrão HL7v2;
- *Sistema B* (6): utiliza o padrão FHIR, o qual é a própria RNDS;
- Mensagem enviada: é apresentada na Figura 3.13;
- *URI* de chamada para transcodificar *hl7v2* em *FHIR* e enviar para a RNDS:

  transcoder_server.com/hl7v2/RNDS.

O fluxo detalhado é apresentado no diagrama de sequência da Figura 3.12, esse fluxo é apresentado pelos passos enumerados na mesma Figura.

Passo 1 *Solicita Transcodificação*: este passo é o que ativa o sistema de transcodificação. Ou seja, o *Sistema A* solicita a transcodificação enviando uma requisição para o *Servidor de borda ou nuvem* (1) da Figura 3.10, e a mensagem da requisição é primeiramente processada pelo *Serviço Ingress* (2) da Figura 3.10.

Passo 2 é verificada se o usuário que está acessando pelo *Sistema A* é autenticado. Caso o usuário não seja autenticado, o fluxo segue para o passo 11 da Figura 3.12, e retorna mensagem de erro ao usuário.

Passo 3 é feita a leitura de qual é o servidor ou cliente de destino e qual o formato que foi recebido, por exemplo, uma requisição HTTP pode ser realizada para o endereço *URI*: *transcoder_server.com/hl7v2/RNDS*. Nesse exemplo, o endereço do servidor de transcodificação (*URL*) é transcoder_server.com, recebendo como informações adicionais:

- */hl7v2*: indica que está recebendo uma mensagem do padrão HL7v2;
- */RNDS*: assinala ao servidor que o destino da mensagem é a RNDS, como esse servidor é FHIR, então terá que transcodificar a mensagem HL7v2 em FHIR.

Em seguida, as informações são enviadas para o *receptor*. Na condição do *receptor* não estar disponível ou não ter sido implementado, é executado o passo 11 da Figura 3.12, retornando uma mensagem de erro para o *Sistema A*.

Passo 4 no receptor, a mensagem chega no módulo *A_to_B* (31) da Figura 3.10. Nesse módulo (31), é realizada a verificação da mensagem pelo *Verificador e validador de dados recebidos* (311) utilizando as *Bibliotecas de interpretação* (312), isto é, verifica se a mensagem é do formato HL7 v2, se existem informações fora do formato e se estão faltando informações obrigatórias. Caso a mensagem recebida seja válida, a informação é enviada para o *Solicitador de transcodificador* (313), então é processado qual o tipo de recurso foi enviado, neste exemplo é apresentada a informação "*VXU_V04*" sublinhada na Figura 3.13. Essa informação sublinhada contém a informação que a mensagem é uma atualização não solicitada da carteira de vacinação do paciente. Dessa forma, caso exista o recurso de mapeamento de "*VXU_V04*" para o "*Bundle*" do FHIR, então é solicitada a transcodificação, ou seja, os dados de saúde recebidos são enviados para o respectivo módulo no *Recursos A_B* (33), então podemos renomear o recurso n-ésimo para *ResAn_VacinationBn* (34) neste exemplo. No caso de não existir o recurso de transcodificação ou o mesmo não estiver disponível, o passo 10 da Figura 3.12 é realizando, e envia uma mensagem de erro para o usuário.

Passo 5 A estrutura interna do *ResAn_VacinationBn* (34) é apresentada na Figura 3.11, dessa forma, os dados são recebidos e transformados no *Transformador de dados parar estrutura de dados interna* (341) utilizando as *Bibliotecas de interpretação fonte* (340). Com essa transformação, os dados são adequados pelo *Ajustador de dados* (342), isto é, são adicionadas informações sobre o padrão que são necessárias, mas não existem na mensagem inicial, e alterar os dados para unidades de medidas caso necessário. Com esses dados no dicionário é possível processá-los pelo *Transcodificador de dados de segmentos de dados* (344), e utilizando as *bibliotecas de mapeamento destino* (343) obtém-se os dados no formato alvo, que neste exemplo é o FHIR. Neste ponto, o corpo da requisição é enviado para o módulo *Montador de requisições* (345). Nessa última etapa, a requisição no formato alvo é montada e enviada para o *Serviço Egress* (4).

Passo 6 no *Serviço Egress* (4) da Figura 3.11 é verificado se o módulo interno que está realizando a requisição para fora do *Servidor de borda* (1) tem a devida permissão para realizar a requisição. Caso não tenha a permissão, o passo 8 da Figura 3.12 é acionado e retorna uma mensagem de erro indicando que o *Sistema A* não tem acesso ao destino.

Passo 7 é executado quando o Passo 6 envia a mensagem com sucesso, por exemplo, a

resposta apresentada na Figura 3.9 ao destino *Sistema B* (6). Em seguida, os passos 8, 9, 10 e 11 são executados em cadeia para responder ao usuário que ocorreu um sucesso na requisição.

Figura 3.13: Exemplo de JSON contendo dados HL7 v2 enviados para o transcodificador.

```
{
    "message": [
        "MSH|^~\\&|NISTEHRAPP|NISTEHRFAC|NISTIISAPP|NISTIISFAC|
            20150624084727.655-0500||VXU^V04^VXU_V04|NIST-IZ-AD-2.
            1_Send_V04_Z22|P|2.5.1|||ER|AL|||||Z22^CDCPHINVS|
            NISTEHRFAC|NISTIISFAC",
        "PID|1||90012^^^NIST-MPI-1^MR||Wong^Elise^^^^^L||19830615|F|
            |2028-9^Asian^CDCREC|9200 Wellington
            Trail^^Bozeman^MT^59715^USA^P||
            ^PRN^PH^^^406^5557896~^NET^^Elise.Wong@isp.com||||||||
            2186-5^Not Hispanic or Latino^CDCREC||N|1|||||N",
        "PD1||||||||||||02^Reminder/recall - any method^HL70215|N|
            20150624|||A|19830615|20150624"
    ]
}
```

Fonte: Adaptada de [47]

3.6 Resumo do Capítulo

Neste capítulo foi apresentado o modelo arquitetural desenvolvido para a solução do problema de engenharia proposto. Inicialmente foi feita a apresentação do fluxograma da arquitetura geral, o qual indica o funcionamento do sistema em alto nível. Em seguida foi apresentada a arquitetura, explicando a função de cada componente e seus módulos internos. Ao final, foi apresentado o fluxo detalhado do funcionamento da arquitetura.

Capítulo 4

Implementação e Resultados Experimentais

Neste capítulo, apresenta-se a implementação da arquitetura de microsserviços para transcodificação de dados de saúde e os resultados de experimentos de validação. A implementação foi realizada utilizando os padrões C-CDA e HL7 v2 transcodificados para FHIR. Na seção 4.1 é apresentada a plataforma utilizada nos experimentos. Na seção 4.2 são apresentados os resultados dos experimentos. Na seção 4.2.1 é apresentado o experimento de *Escalabilidade de Transcodificadores* para validar a premissa (b) *escalabilidade* da seção 3.1. Na seção 4.2.2 apresenta-se o experimento de *Reconfiguração de Serviços*, o qual surge da necessidade da premissa (c) *flexibilidade de escolha de padrões* da Seção 3.1. Na seção 4.2.3 é apresentado o experimento de *Inserção de Diferentes Provedores de Saúde* para a premissa (d) *flexibilidade de inserir novos provedores de saúde* da Seção 3.1.

4.1 Plataforma do Experimento

Nesta seção é apresentada a plataforma utilizada nos experimentos, contendo as informações dos materiais, software e como foi realizar a base da implementação.

A base da plataforma é o *Istio*, representado pelo (1) da Figura 3.4, a qual é a *malha de serviço* utilizada para orquestrar e observar a aplicação em microsserviços. Nele são definidos:

- *Servidor* (1 da Figura 3.4): a plataforma é configurada e executada em um computador comercial, processador intel core i7 8700, 32 GB de memória ram 2400 MHz, 1 TB de Hdd.

- *Serviço Ingress* (2 da Figura 3.4): a configuração do *Serviço Ingress* foi feita para receber mensagens do tipo *Hl7v2* e *CCDA*. Sendo as mesmas recebidas pelo servidor nos respectivos caminhos de requisições "localhost/hl7" e "localhost/ccda";

- *Serviço Egress* (4 da Figura 3.4): a configuração do *Serviço Egress* foi mantida padrão, ou seja, permitindo que os serviços de dentro da plataforma *Servidor de Borda* (1 da Figura 3.4) enviem requisições para outros servidores. Por exemplo, enviar a mensagem transformada para um servidor FHIR destino;

- *Software* utilizados:

  – Docker[1], programa de execução de contêiners;

  – Istio, *malha de serviço* utilizada para os microsserviços.

Figura 4.1: Seleção de receptores(30) no Ingress

```
http:
- match:
  - uri:
      exact: /ccda
  - uri:
      prefix: /ccda
http:
- match:
  - uri:
      exact: /hl7
  - uri:
      prefix: /hl7
```

Fonte: Elaborada pelo autor.

Inicialmente é realizada a configuração da *malha de serviço* Istio[2]. Nesta etapa, serão

[1] https://www.docker.com/
[2] https://istio.io/latest/docs/setup/getting-started/

geradas as configurações-base da plataforma, como ferramentas de observabilidade, *Serviço Ingress* e *Serviço Egress* padrão.

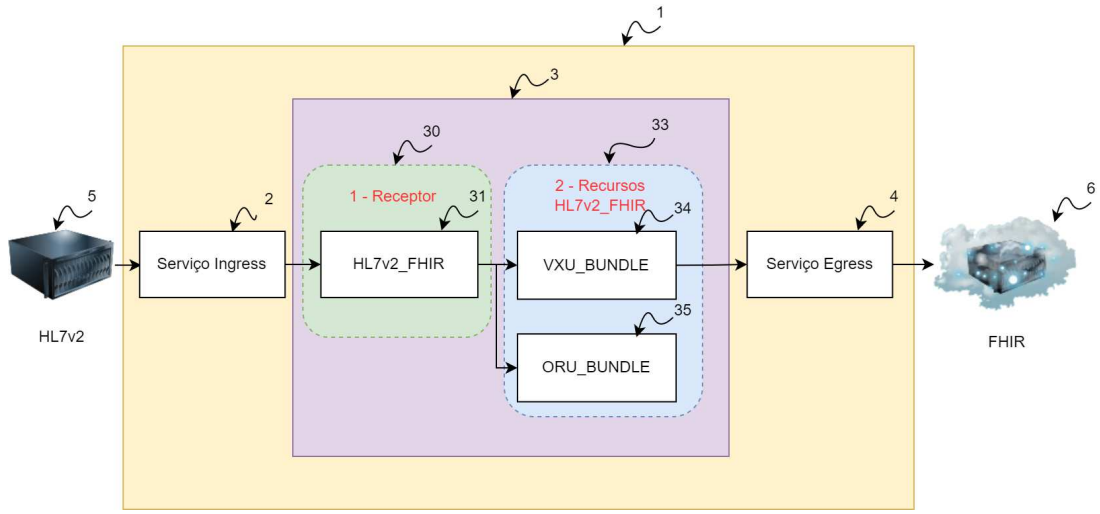
Devido à utilização dos padrões de saúde HL7 v2 e C-CDA em transcodificadores como o projeto *FHIR-Converter* [15], esses padrões foram adotados, pois é possível comparar os resultados e ajustar o transcodificador implementado. Ou seja, segue padrão de transcodificadores já existentes.

4.1.1 HL7v2 para FHIR

A configuração da plataforma para o funcionamento do transcodificador HL7v2 para FHIR é feita da seguinte forma, utilizando os módulos apresentados na Figura 4.2:

- é adicionado o seletor de HL7v2 para FHIR no *Ingress* (2);
- o serviço Receptor *HL7v2_FHIR* (31) é inicializado;
- são inicializados os serviços de Recursos *VXU_BUNDLE* (34) e ORU_BUNDLE (35).

Figura 4.2: Diagrama de transcodificadores HL7v2 para FHIR.



Fonte: Elaborada pelo autor.

Com o *Serviço Ingress* (2) configurado, conforme apresentado na Figura 4.2, quando o usuário enviar uma requisição para o *Servidor* (1), essa mensagem é então redirecionada, pelo *Serviço Ingress* (2), para o serviço de receptor *hl7v2_FHIR* (31). A seleção da chamada

é realizado pelo caminho da requisição para *localhost/hl7v2*, e esse caminho indica uma transcodificação de HL7v2 para FHIR. A seleção do *Receptor* (30) é feita pelo *Serviço Ingress* (2) que realiza esse roteamento da requisição para o *Receptor* (30) correto.

A implementação dos transcodificadores foi feita em *Python*. A implementação utiliza a biblioteca "*hl7*"[3]. Essa biblioteca provê funções para validação e leitura dos dados do formato HL7v2. A implementação foi realizada seguindo os modelos dos algoritmos 1 e 2 da Seção 3.4. Dessa forma, o primeiro passo foi utilizar a biblioteca *hl7* para validar os dados enviados pelo usuário são válidos.

Após a validação, é feita a leitura de qual o tipo de recurso, por exemplo, é possível verificar se o recurso é do tipo "VXU"(esse recurso é uma atualização de registro de prontuário de vacina não solicitado).

Em seguida, é realizada a requisição para o serviço do recurso correspondente, por exemplo, para o recurso do módulo *VXU_BUNDLE* (34) apresentado na Figura 4.2. Caso o recurso esteja indisponível na requisição, o recurso responde com erro ou a requisição excede o tempo limite de espera. Caso o recurso esteja disponível, o mesmo deverá responder com a mensagem transcodificada.

Quando o recurso *VXU_BUNDLE*, apresentado na Figura 4.2, recebe a mensagem, o mesmo utiliza a biblioteca "hl7" para acessar os dados da estrutura do padrão HL7v2 e transcodifica para a estrutura do FHIR.

Após transcodificar a mensagem e montar a mensagem no formato alvo, é realizado o envio da mensagem transcodificada para o sistema de destino.

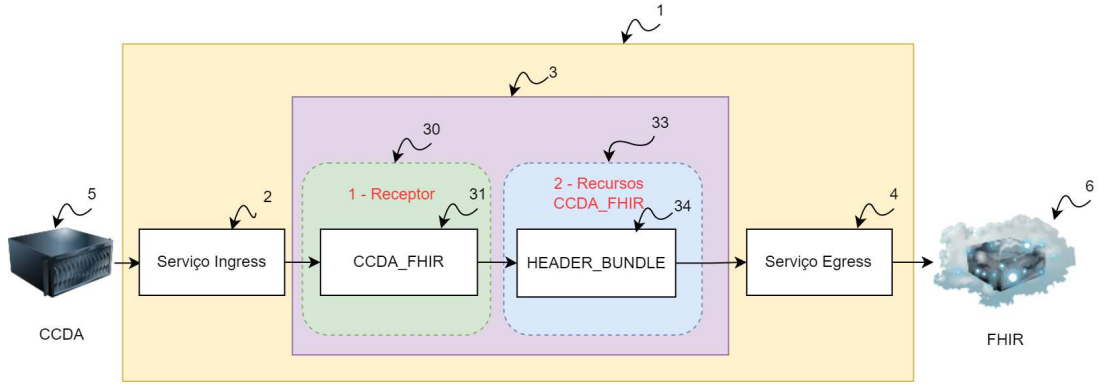
4.1.2 CCDA para FHIR

A configuração da plataforma para o funcionamento do transcodificador CCDA para FHIR é feita da seguinte forma, utilizando os módulos apresentados na Figura 4.3:

- o seletor de CCDA para FHIR é adicionado no *Ingress* (2);
- o serviço Receptor *CCDA_FHIR* (31) é inicializado;
- é inicializado o serviço de Recurso *HEADER_BUNDLE* (34).

[3] https://python-hl7.readthedocs.io/en/latest/

Figura 4.3: Diagrama de transcodificador CCDA para FHIR.



Fonte: Elaborada pelo autor.

O transcodificador CCDA para FHIR é apresentado na Figura 4.3. Quando o usuário envia uma requisição para o *Servidor de Borda* (1), essa mensagem é redirecionada para o *CCDA_FHIR* (31), devido à seleção realizada pelo *Serviço Ingress* (2).

A implementação deste transcodificador foi realizada em *Python*. Devido à utilização do formato *XML* do CCDA, foi utilizada a biblioteca padrão de *XML*[4] do *Python*. Pois essa biblioteca provê mecanismos de leitura da estrutura de dados *XML* e validação se o mesmo não está corrompido.

Após a validar os dados de entrada, os mesmos são enviados para o recurso *HEADER_BUNDLE* (34) apresentado na Figura 4.3. Para simplificação, outros recursos de CCDA não foram implementados, mas seguem a mesma lógica de implementação deste.

Quando a mensagem chega no módulo *HEADER_BUNDLE* (34) da Figura 4.3, os dados dessa mensagem são lidos e ajustados para o formato do padrão de saúde alvo, FHIR neste caso. Esse ajuste pode ser, por exemplo, a transformação do formato data do CCDA "20050329171504+0500" em FHIR "2005-03-29T09:15:04-03:00".

Após realizar esses ajustes, a mensagem transcodificada é inserida no formato da mensagem do padrão alvo, além de adicionar outras informações adicionais que o padrão de saúde alvo precisa ter em sua mensagem, mas não existe no padrão de saúde que está sendo transcodificado.

Com a mensagem pronta no formato do padrão de saúde alvo, o transcodificar faz a requisição para o servidor/cliente alvo, que neste exemplo é o FHIR (6) da Figura 4.3.

[4] https://docs.python.org/3/library/xml.etree.elementtree.html

4.2 Resultados Experimentais

Nesta seção os experimentos relacionados a arquitetura proposta são apresentados e discutidos. Na seção 4.2.1 é apresentado o experimento de escalabilidade, onde é apresentada a funcionalidade de escalar a arquitetura com a eficiência de utilização de recursos de microsserviços, este experimento é validado em relação aos requisitos de QoS em saúde apresentados na Tabela 3.1. A seção 4.2.2 contém o experimento de reconfiguração dos serviços, sendo demonstrada a viabilidade dessa funcionalidade em tempo de execução de serviço. Na seção 4.2.3 é apresentado o experimento de inserção de novos provedores de saúde que irão se comunicar utilizando a aplicação.

4.2.1 Escalabilidade de Transcodificadores

Transcodificadores têm função essencial no sistema, ou seja, caso os mesmos não estejam em execução, a comunicação entre sistemas com padrões diferentes fica impossibilitada. Esse cenário é representado na Figura 4.4, pois, o *Sistema 1 HL7v2* (1) consegue se comunicar diretamente com *Sistema 2 HL7v2* (2), mas o *Sistema 1 HL7v2* (1) não consegue trocar informações com o *Sistema 2 FHIR* (4) diretamente. Nesse caso, pode-se inserir o *Transcodificador HL7v2 <=> FHIR* (5) entre os sistemas (1) e (4) para possibilitar a comunicação entre eles.

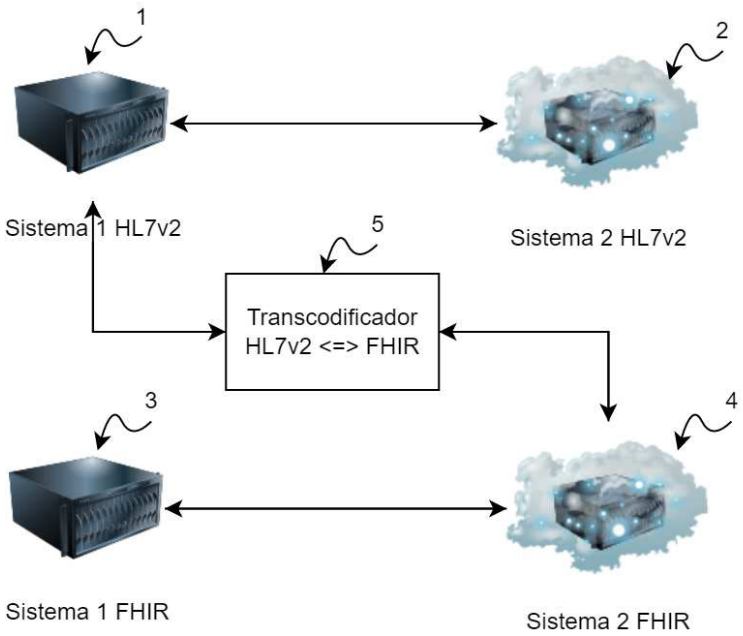
A escalabilidade de microsserviços, nesse contexto, traz algumas vantagens em relação a sistemas monolíticos:

- reconfiguração sem reiniciar: em sistemas monolíticos, caso precise reconfigurar um serviço ou alterar a versão de implementação, é preciso reiniciar o sistema, mas no caso de microsserviços, é possível inicializar a execução de cópias do novo serviço e ir removendo da execução os serviços antigos, assim não é preciso reiniciar o sistema, nem interromper o seu uso;

- tempo de resposta: para manter a qualidade de serviço, o sistema pode escalar, mantendo assim o tempo de resposta em níveis aceitáveis;

- aumento de confiabilidade do sistema: devido à essencialidade dos transcodificadores na comunicação, o uso de cópias dos microsserviços aumenta a confiabilidade do sis-

Figura 4.4: Comunicação entre sistemas Hl7v2 e FHIR.



Fonte: Elaborada pelo autor.

tema via redundância, o qual é um ponto de extrema importância como apresentado por Nunes et al. em [12].

O objetivo deste experimento é demonstrar ser possível reconfigurar a plataforma sem precisar reiniciá-la e demonstrar que, ao escalar, o tempo de resposta diminui em função da quantidade de requisições realizadas. O ambiente utilizado neste experimento é apresentado na Seção 4.1.2.

As requisições deste experimento foram realizadas paralelamente, utilizando programação concorrente em *Python*. É apresentado na Tabela 4.1, o tempo de resposta de requisições para a implementação do CCDA de somente uma cópia, ou seja, essa será a referência funcionando como se fosse um sistema monolítico. Pois, se fosse utilizado outro sistema implementado, haveria diferença no tempo de resposta devido à implementação, linguagem e ambiente. Mas, utilizando o mesmo serviço, é possível mensurar o quão eficiente é a diminuição do tempo de resposta do mesmo, quando se aumenta as cópias do serviço. Dessa forma, também é apresentada a Tabela 4.2, a qual contém o tempo de resposta para três testes feitos utilizando cinco cópias do serviço de transcodificação CCDA. Comparando as duas Tabelas, fica evidente que ao utilizar mais cópias de microsserviços, menor é o

Tabela 4.1: Testes de tempo de resposta para 1 cópia (monolítico).

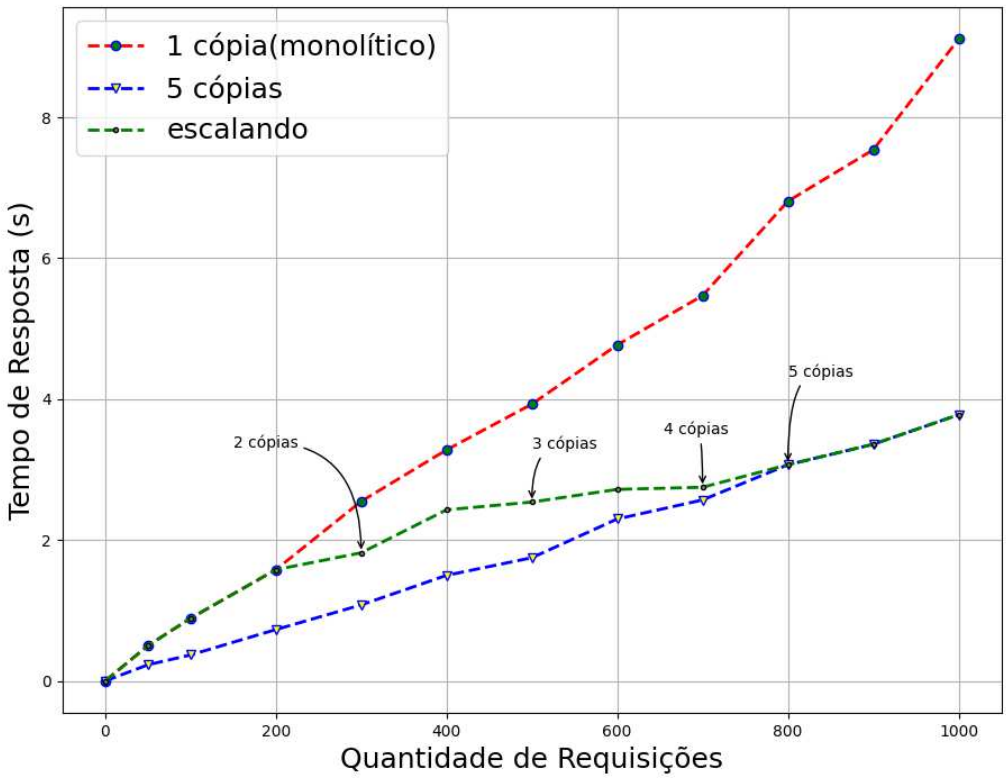
Quantidade de requisições	Teste A (s)	Teste B (s)	Teste C (s)	Média (s)
50	0,46	0,58	0,47	0,50
100	0,84	1,08	0,74	0,89
200	1,70	1,53	2,67	1,58
300	2,48	2,49	2,67	2,55
400	3,02	3,84	2,97	3,28
500	3,68	4,30	3,80	3,93
600	4,65	5,11	4,54	4,77
700	5,32	5,71	5,39	5,47
800	6,84	6,93	6,65	6,81
900	7,88	7,67	7,06	7,54
1000	9,97	8,53	8,84	9,11
10000	81,81	84,44	80,75	82,33

Tabela 4.2: Testes de tempo de resposta para 5 cópias do transcodificador.

Quantidade de requisições	Teste A (s)	Teste B (s)	Teste C (s)	Média (s)
50	0,24	0,25	0,21	0,23
100	0,35	0,43	0,33	0,37
200	0,68	0,70	0,82	0,73
300	1,04	1,04	1,16	1,08
400	1,56	1,44	1,49	1,50
500	1,73	1,84	1,69	1,75
600	2,60	2,31	1,98	2,30
700	2,40	2,59	2,74	2,57
800	3,64	2,75	2,82	3,07
900	3,00	3,68	3,39	3,36
1000	3,38	4,08	3,87	3,78
10000	39,25	48,08	48,54	45,29

tempo de resposta. Testes feitos com mais cópias não foram possíveis devido à limitação do computador utilizado.

Figura 4.5: Comparação do tempo de resposta devido à escalabilidade.



Fonte: Elaborada pelo autor.

Os testes de medição de tempo de resposta representados na Figura 4.5, mostraram que a escalabilidade serve também para manter a qualidade de serviço do serviço de Alarmes apresentada na Tabela 3.1. Pois, em um servidor, basta escalar a depender da carga a qual está submetido, conseguindo assim, manter o sistema funcionando eficientemente.

Neste experimento, também foram geradas cópias do microsserviço de transcodificação, além das cinco utilizadas anteriormente, sempre iniciando com nenhuma cópia e escalando até o número de cópias apresentado na Tabela 4.3. O tempo foi medido pela diferença de tempo da criação da primeira cópia e a inicialização da última cópia. Demonstrando que o sistema consegue, de forma rápida e automática, suprir um aumento na carga dos serviços,

Tabela 4.3: Tempo necessário para aumentar quantidade de serviços iniciando com uma cópia.

Quantidade de cópias geradas	Tempo em segundos.
5	1,46
10	4,76
15	6,65
20	8,82
25	12,1
30	14,25

em tempo de execução.

4.2.2 Reconfiguração de Serviços

Devido aos padrões de saúde serem atualizados com o tempo e novos serem criados, se faz necessário ter uma forma de, não somente implementar, mas também de inserir na plataforma esses transcodificadores, tornando a plataforma um eixo central na comunicação entre os sistemas e provendo assim interoperabilidade. Esse experimento é devido à premissa (c)*flexibilidade de escolha de padrões* da Seção 3.1, comprovando a alta capacidade de reconfiguração da plataforma.

A reconfiguração para adicionar um transcodificador precisa de duas etapas: adicionar ao *Serviço Ingress* a configuração do caminho do transcodificador, como apresentada na Figura 4.1, e inicializar o serviço de transcodificação. Exemplo de configuração do transcodificador CCDA para FHIR:

- insere a configuração do caminho de requisição para o CCDA no *Serviço Ingress*, bastando executar o comando de aplicar o *Serviço Ingress* no terminal;
- inicializa os serviços de receptores e transcodificação na plataforma executando o comando de aplicar os serviços.

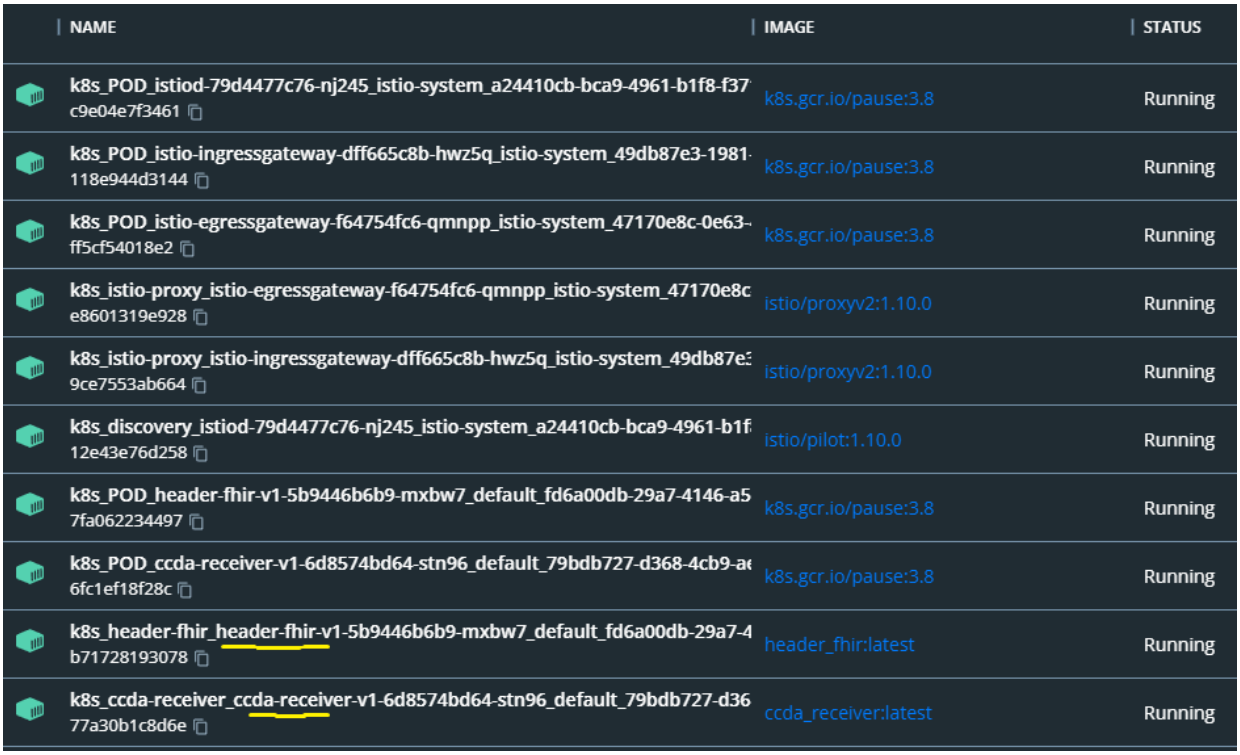
O objetivo deste experimento é comprovar ser possível reconfigurar a plataforma adicionando um novo transcodificador, com a vantagem de não precisar reiniciar a plataforma. A configuração da plataforma utilizada é apresentada na Seção 4.1.

O teste foi feito executando os comandos da plataforma para inserir o CCDA para FHIR. Dessa forma, os serviços de transcodificação já estão em execução conforme os serviços

grifados na Figura 4.6. Na mesma Figura estão assinalados, em amarelo, os serviços de receptor(*ccda-receiver-v1*) e transcodificação (*header-fhir-v1*) que foram configurados e estão em execução.

Para realizar reconfigurações, por exemplo, escalar a quantidade de transcodificadores da Seção 4.2.1, basta alterar a quantidade de transcodificadores do arquivo de configuração do serviço e executar novamente o seguinte comando de aplicar.

Figura 4.6: Serviços disponíveis no contêiner



Fonte: Elaborada pelo autor.

Este experimento cumpre a premissa (c) *flexibilidade de escolha de padrões* da Seção 3.1, pois com a simplicidade de dois comandos, é possível adicionar ou remover um tipo de transcodificador da plataforma de transcodificadores.

4.2.3 Inserção de Diferentes Provedores de Saúde

Nesta seção, é apresentada a inserção de diferentes provedores de saúde, tanto como solicitantes de transcodificação, quanto recebendo mensagens transcodificadas.

A possibilidade de adicionar novos provedores à plataforma é necessária em situações,

por exemplo, de pandemias em que se faz necessária a troca de informações entre vários provedores de saúde [48]. Outra situação em que é necessário ajustar também para vincular a um barramento de saúde, por exemplo, à RNDS[5], para ter os dados acessíveis de múltiplas localidades e por vários provedores de saúde.

Inserção de Provedor de Saúde Solicitante de Transcodificação

Neste experimento foi configurada a política de autorização que determina quais endereços *ip* podem se comunicar solicitando transcodificação na plataforma. Apresentado na Figura 4.7 sublinhado de vermelho é o endereço *ip* que está permitido, o exemplo contém o *ip* "1.2.3.4", ou seja, qualquer outro *ip* que tentar se comunicar receberá uma mensagem de erro "*RBAC: access denied*" e código HTTP 403 indicando que o acesso é proibido. Para adicionar um novo provedor de saúde à rede, deve-se inserir o *ip* na lista e executar o comando de reconfiguração do *Serviço Ingress*.

Figura 4.7: Serviço de política de autorização

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: ingress-policy
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  action: ALLOW
  rules:
  - from:
    - source:
        ipBlocks: ["1.2.3.4"]
```

Fonte: Elaborada pelo autor.

[5] https://www.gov.br/saude/pt-br/assuntos/rnds

Inserção de Provedor de Saúde Destinatário de Transcodificação

A solução proposta define o destino como sendo uma informação a mais enviada na requisição, ou seja, a requisição será feita para "*http://localhost/ccda/local_fhir*", onde *local_fhir* é o nome do servidor FHIR para onde a mensagem será enviada. Para adicionar ao código é preciso realizar os seguintes passos:

- inserir o caminho de acesso no *Serviço Ingress*, conforme apresentado na Figura 4.8. Dessa forma, a requisição aceita o servidor destino *local_fhir* na requisição;
- para configurar o *ip* e a porta do servidor destino, deve-se adicionar essas informações no serviço dos transcodificadores, por exemplo, *HEADER_BUNDLE* (34) da Figura 4.3, que contém as configurações dos servidores destino, conforme apresentado nas Figuras 4.9 e 4.10.

Após essa configuração, fica disponível a seguinte requisição: "*localhost/ccda/local_fhir*" para os provedores de saúde aceitos no *Serviço Ingress*. Então, é possível enviar mensagens transcodificadas para o servidor *local_fhir*.

Figura 4.8: Adição do caminho de destino aceito no Ingress.

```
- uri:
    exact: /ccda/local_fhir
```

Fonte: Elaborada pelo autor.

Figura 4.9: Adição do *ip* do destino.

```
- name: LOCAL_FHIR_SERVER
  value: "localhost"
- name: LOCAL_FHIR_SERVER_PORT
  value: "8080"
```

Fonte: Elaborada pelo autor.

Este experimento de adicionar provedor destinatário conseguiu cumprir a sua função de inserir novos destinatários para a transcodificação. Permitindo assim, uma configuração flexível da plataforma, devido à arquitetura utilizada. Esse experimento demonstrou ser

Figura 4.10: Adição do *ip* de destino no montador de requisições.

```
LOCAL_FHIR_SERVER = os.getenv("LOCAL_FHIR_SERVER", "localhost")
LOCAL_FHIR_SERVER_PORT = os.getenv("LOCAL_FHIR_SERVER_PORT", "8080")
if(server == 'local_fhir'):
    Fhir_server = 'http://' + LOCAL_FHIR_SERVER + ':' + LOCAL_FHIR_SERVER_PORT + '/fhir'
    headers = {'Host': LOCAL_FHIR_SERVER+':'+LOCAL_FHIR_SERVER_PORT, 'Content-Type':'application/json'}
```

Fonte: Elaborada pelo autor.

possível reconfigurar os serviços mantendo a segurança proveniente da *malha de serviço*, e como demonstrado no experimento de escalabilidade, o sistema não precisa ser reiniciado.

4.3 Resumo do Capítulo

Neste capítulo a plataforma de transcodificação foi descrita, considerando as premissas apresentadas na Seção 3.1. Foram detalhados os algoritmos utilizados para a implementação dos transcodificadores, destacando suas funcionalidades e o que as motiva.

O experimento de *Escalabilidade de Transcodificadores* foi bem-sucedido, pois como apresentado na Figura 4.5, o aumento da quantidade de cópias dos microsserviços ajudam a manter o tempo de resposta menor que o limite de qualidade de serviço de saúde. Dessa forma, conforme aumentar a quantidade de cópias, melhor será o tempo de resposta para uma dada quantidade de requisições.

O experimento de *Reconfiguração de Serviços* provou ser possível reconfigurar a plataforma, sem impactar nos serviços que estão em funcionamento. Ou seja, é possível inserir novos transcodificadores, remover antigos e atualizar transcodificadores, mantendo o sistema em funcionamento.

O experimento de *Inserção de Diferentes Provedores de Saúde* demonstrou que existem mecanismos de inserir novos provedores na configuração da *malha de serviço*. Mantendo assim, a segurança dos dados dos pacientes.

Capítulo 5

Considerações Finais

Este capítulo contém o resumo das contribuições do trabalho e as conclusões obtidas do mesmo. Na Seção 5.1 são apresentadas as conclusões do trabalho, e na Seção 5.2 é indicada a possibilidade de trabalho futuro.

5.1 Conclusões

Nesta dissertação, foram realizadas investigações de soluções para prover interoperabilidade, e que a partir de experimentos foi identificada a viabilidade do uso de transcodificador implementado em microsserviços, assim como, o uso do FHIR como padrão de saúde base, devido à sua crescente difusão. Dessa forma, foi apresentada uma proposta de arquitetura em microsserviços para transcodificadores de padrões de saúde.

Os resultados experimentais demonstraram que a solução arquitetural cumpre as premissas apresentadas no Capítulo 3 de: transcodificar os dados (a), escalabilidade horizontal eficiente (b), flexibilidade de escolha de padrões (c), flexibilidade de inserir novos provedores de saúde (d). A escalabilidade advinda dos microsserviços auxilia em garantir que os requisitos de Qualidade de Serviço, como os apresentados na Tabela 3.1, utilizando eficientemente os recursos computacionais. O experimento de escalabilidade demonstrou que ao adicionar novas cópias do serviço de transcodificação, foi possível manter dentro dos limites de qualidade de serviço para mais requisições de *Alarme* realizadas simultaneamente.

Foi possível demonstrar que é viável encapsular funções de transcodificação de modo

independente, permitindo que elas possam se beneficiar das características do modelo arquitetural de microsserviços.

A arquitetura também demonstrou sua capacidade de realizar a transcodificação e prover interoperabilidade para sistemas de saúde, que se apresenta ainda hoje como um problema na utilização e acesso aos dados de saúde.

5.2 Propostas para Trabalhos Futuros

Como sugestões para trabalhos futuros, destacam-se:

- levantar os requisitos de qualidade de serviço de saúde e criar uma base de dados em um padrão de saúde, por exemplo, FHIR. Utilizando essa base de dados, criar uma ferramenta de validação de transcodificadores, fazendo as chamadas para o transcodificador e verificando se o tempo de resposta corresponde com os requisitos de QoS e se a transcodificação foi realizada corretamente;

- implementar funcionalidade nos transcodificadores para solicitarem, aos servidores de saúde conectados, atualizações das informações de pacientes, e enviarem para o barramento de saúde, por exemplo, a RNDS.

Referências Bibliográficas

1 ALMOHAMMADI, Nawal; SEN, Adnan Ahmed Abi; BORIE, Hussein; ALMUHAMMADI, Abdullah; ALKHODRE, Ahmad; YAMIN, Mohammad. A framework for enhancing relief system of health domain by iot. In: *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)*. [S.l.: s.n.], 2019. p. 1326–1330.

2 AL-MARRIDI, Abeer Z.; MOHAMED, Amr; ERBAD, Aiman; AL-ALI, Abdulla; GUIZANI, Mohsen. Efficient eeg mobile edge computing and optimal resource allocation for smart health applications. In: *2019 15th International Wireless Communications Mobile Computing Conference (IWCMC)*. [S.l.: s.n.], 2019. p. 1261–1266.

3 SARIPALLE, Rishi; RUNYAN, Christopher; RUSSELL, Mitchell. Using hl7 fhir to achieve interoperability in patient health record. *Journal of Biomedical Informatics*, v. 94, p. 103188, 2019. ISSN 1532-0464. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1532046419301066>.

4 LEHNE, Moritz; SASS, Julian; ESSENWANGER, Andrea; SCHEPERS, Josef; THUN, Sylvia. Why digital medicine depends on interoperability. *npj Digital Medicine*, v. 2, n. 1, 2019.

5 NOURA, Mahda; ATIQUZZAMAN, Mohammed; GAEDKE, Martin. Interoperability in internet of things: Taxonomies and open challenges. *Mobile Networks and Applications*, v. 24, n. 3, p. 796–809, 2018.

6 HL7. *FHIR*. 2019. Acessado em: 06/07/2021. Disponível em: <https://www.hl7.org/fhir/overview.html>.

7 Borisov, V.; Minin, A.; Basko, V.; Syskov, A. Fhir data model for intelligent multimodal interface. In: *2018 26th Telecommunications Forum (TELFOR)*. [S.l.: s.n.], 2018. p. 420–425.

8 SAÚDE, Ministério da. *FASES DO PROJETO*. 2020. Acessado em: 01/07/2021. Disponível em: <https://rnds.saude.gov.br/fases-do-projeto/>.

9 IEEE Health informatics–PoC medical device communication Part 00101: Guide–Guidelines for the use of RF wireless technology. *IEEE Std 11073-00101-2008*, p. 1–125, 2008.

10 Shoumik, F. S.; Talukder, M. I. M. M.; Jami, A. I.; Protik, N. W.; Hoque, M. M. Scalable micro-service based approach to fhir server with golang and no-sql. In: *2017 20th*

International Conference of Computer and Information Technology (ICCIT). [S.l.: s.n.], 2017. p. 1–6.

11 ASSUNÇÃO, Wesley K. G.; KRÜGER, Jacob; MENDONÇA, Willian D. F. Variability management meets microservices: Six challenges of re-engineering microservice-based webshops. In: *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A*. New York, NY, USA: Association for Computing Machinery, 2020. (SPLC '20). ISBN 9781450375696. Disponível em: <https://doi.org/10.1145/3382025.3414942>.

12 LIU, Zheng; YU, Huiqun; FAN, Guisheng; CHEN, Liqiong. Reliability modeling and analysis of hospital information system based on microservices. In: *2021 IEEE International Conference on Progress in Informatics and Computing (PIC)*. [S.l.: s.n.], 2021. p. 313–318.

13 BETTONI, Giovani Nícolas; LOBO, Thafarel Camargo; FLORES, Cecília Dias; SANTOS, Bruno Gomes Tavares dos; SILVA, Filipe Santana da. Application of hl7 fhir in a microservice architecture for patient navigation on registration and appointments. In: *2021 IEEE/ACM 3rd International Workshop on Software Engineering for Healthcare (SEH)*. [S.l.: s.n.], 2021. p. 44–51.

14 WIERINGA, Roel J. *Design science methodology for information systems and software engineering*. [S.l.]: Springer, 2014.

15 MICROSOFT. *FHIR-Converter*. 2022. Disponível em: <https://github.com/microsoft/FHIR-Converter>.

16 NAVEED, Arjmand; SIGWELE, T; HU, Yim Fun; KAMALA, M; SUSANTO, Misfa. Addressing semantic interoperability, privacy and security concerns in electronic health records. *Journal of Engineering and Scientific Research*, v. 2, n. 1, Jun 2020.

17 GANSEL, Xavier; MARY, Melissa; BELKUM, Alex Van. Semantic data interoperability, digital medicine, and e-health in infectious disease management: a review. *European Journal of Clinical Microbiology & Infectious Diseases*, v. 38, n. 6, p. 1023–1034, Jun 2019.

18 HIMSS. *Himss*. 2021. Acessado em: 30/06/2021. Disponível em: <https://www.himss.org/resources/interoperability-healthcare>.

19 RANGANATHAN, Parthasarathy; STODOLSKY, Daniel; CALOW, Jeff; DORFMAN, Jeremy; GUEVARA, Marisabel; IV, Clinton Wills Smullen; KUUSELA, Aki; BALASUBRAMANIAN, Raghu; BHATIA, Sandeep; CHAUHAN, Prakash; CHEUNG, Anna; CHONG, In Suk; DASHARATHI, Niranjani; FENG, Jia; FOSCO, Brian; FOSS, Samuel; GELB, Ben; GWIN, Sara J.; HASE, Yoshiaki; HE, Da-ke; HO, C. Richard; JR., Roy W. Huffman; INDUPALLI, Elisha; JAYARAM, Indira; KONGETIRA, Poonacha; KYAW, Cho Mon; LAURSEN, Aaron; LI, Yuan; LOU, Fong; LUCKE, Kyle A.; MAANINEN, JP; MACIAS, Ramon; MAHONY, Maire; MUNDAY, David Alexander; MUROOR, Srikanth; PENUKONDA, Narayana; PERKINS-ARGUETA, Eric; PERSAUD, Devin; RAMIREZ, Alex; RAUTIO, Ville-Mikko; RIPLEY, Yolanda; SALEK, Amir; SEKAR, Sathish; SOKOLOV, Sergey N.; SPRINGER, Rob; STARK, Don; TAN, Mercedes; WACHSLER, Mark S.;

WALTON, Andrew C.; WICKERAAD, David A.; WIJAYA, Alvin; WU, Hon Kwan. Warehouse-scale video acceleration: Co-design and deployment in the wild. In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: Association for Computing Machinery, 2021. (ASPLOS 2021), p. 600–615. ISBN 9781450383172. Disponível em: <https://doi.org/10.1145/3445814.3446723>.

20 KOZIRI, Maria G.; PAPADOPOULOS, Panos K.; TZIRITAS, Nikos; LOUKOPOULOS, Thanasis; KHAN, Samee U.; ZOMAYA, Albert Y. Efficient cloud provisioning for video transcoding: Review, open challenges and future opportunities. *IEEE Internet Computing*, v. 22, n. 5, p. 46–55, 2018.

21 HOQUE, Mohammad Aminul; HOSSAIN, Mahmud; HASAN, Ragib. Igaas: An iot gateway-as-a-service for on-demand provisioning of iot gateways. In: *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. [S.l.: s.n.], 2020. p. 1–6.

22 BAHRAMI, Mehdi; PARK, Junhee; LIU, Lei; CHEN, Wei-Peng. Api learning: Applying machine learning to manage the rise of api economy. In: *Companion Proceedings of the The Web Conference 2018*. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018. (WWW '18), p. 151–154. ISBN 9781450356404. Disponível em: <https://doi.org/10.1145/3184558.3186966>.

23 THOMAN, Peter; GOGL, Daniel; FAHRINGER, Thomas. Sylkan: Towards a vulkan compute target platform for sycl. In: *International Workshop on OpenCL*. New York, NY, USA: Association for Computing Machinery, 2021. (IWOCL'21). ISBN 9781450390330. Disponível em: <https://doi.org/10.1145/3456669.3456683>.

24 CLEVELAND, Sean B.; JAMTHE, Anagha; PADHY, Smruti; STUBBS, Joe; PACKARD, Michale; LOONEY, Julia; TERRY, Steve; CARDONE, Richard; DAHAN, Maytal; JACOBS, Gwen A. Tapis api development with python: Best practices in scientific rest api implementation: Experience implementing a distributed stream api. In: *Practice and Experience in Advanced Research Computing*. New York, NY, USA: Association for Computing Machinery, 2020. (PEARC '20), p. 181–187. ISBN 9781450366892. Disponível em: <https://doi.org/10.1145/3311790.3396647>.

25 SUTANTO, Jofry H.; SELDON, H. Lee. Translation between hl7 v2.5 and ccr message formats (for communication between hospital and personal health record systems). In: *2011 IEEE Conference on Open Systems*. [S.l.: s.n.], 2011. p. 406–410.

26 LU, Xiaoqi; GU, Yu; ZHAO, Jianfeng; YU, Ning; JIA, Weitao. Research and implementation of medical information format conversion based on hl7 version 2.x. In: *2011 International Conference on Computer Science and Service System (CSSS)*. [S.l.: s.n.], 2011. p. 2440–2443.

27 LEE, Sung-Hyun; SONG, Joon Hyun; KIM, Il Kon. Cda generation and integration for health information exchange based on cloud computing system. *IEEE Transactions on Services Computing*, v. 9, n. 2, p. 241–249, 2016.

28 PETRAKIS, Yannis; KOUROUBALI, Angelina; KATEHAKIS, Dimitrios. A mobile app architecture for accessing emrs using xds and fhir. In: *2019 IEEE 19th International Conference on Bioinformatics and Bioengineering (BIBE)*. [S.l.: s.n.], 2019. p. 278–283.

29 ANDERSEN, Björn; KASPARICK, Martin; ULRICH, Hannes; SCHLICHTING, Stefan; GOLATOWSKI, Frank; TIMMERMANN, Dirk; INGENERF, Josef. Point-of-care medical devices and systems interoperability: A mapping of ice and fhir. In: *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*. [S.l.: s.n.], 2016. p. 1–5.

30 HL7. *FHIR*. 2019. Acessado em: 01/07/2021. Disponível em: <https://www.hl7.org/fhir/patient-example.json.html>.

31 RICHARDSON, Chris. *What are microservices?* 2020. Acessado em: 25/06/2021. Disponível em: <https://microservices.io/>.

32 SINGH, Vindeep; PEDDOJU, Sateesh K. Container-based microservice architecture for cloud applications. In: *2017 International Conference on Computing, Communication and Automation (ICCCA)*. [S.l.: s.n.], 2017. p. 847–852.

33 SUN, Xiaoyan; LIANG, Yun; HUANG, Hui. Design and implementation of internet of things platform based on microservice and lightweight container. In: *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*. [S.l.: s.n.], 2020. v. 9, p. 1353–1357.

34 QIU, Yuqing; LUNG, Chung-Horng; AJILA, Samuel; SRIVASTAVA, Pradeep. Lxc container migration in cloudlets under multipath tcp. In: *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*. [S.l.: s.n.], 2017. v. 2, p. 31–36.

35 SEDGHPOUR, Mohammad Reza Saleh; KLEIN, Cristian; TORDSSON, Johan. Service mesh circuit breaker: From panic button to performance management tool. In: *Proceedings of the 1st Workshop on High Availability and Observability of Cloud Systems*. New York, NY, USA: Association for Computing Machinery, 2021. (HAOC '21), p. 4–10. ISBN 9781450383363. Disponível em: <https://doi.org/10.1145/3447851.3458740>.

36 MENDONçA, Nabor C.; BOX, Craig; MANOLACHE, Costin; RYAN, Louis. The monolith strikes back: Why istio migrated from microservices to a monolithic architecture. *IEEE Software*, v. 38, n. 5, p. 17–22, 2021.

37 SOMASHEKAR, Gagan; GANDHI, Anshul. Towards optimal configuration of microservices. In: *Proceedings of the 1st Workshop on Machine Learning and Systems*. New York, NY, USA: Association for Computing Machinery, 2021. (EuroMLSys '21), p. 7–14. ISBN 9781450382984. Disponível em: <https://doi.org/10.1145/3437984.3458828>.

38 HEALTHCARE, NextGen. *Seamless, Scalable, and Supported Interoperability with NextGen Connect Integration Engine*. 2021. Acessado em: 30/06/2021. Disponível em: <https://www.nextgen.com/products-and-services/integration-engine>.

39 LI, Wei; PARK, JongTae. Design and implementation of integration architecture of iso 11073 dim with fhir resources using coap. In: *2017 International Conference on Information and Communications (ICIC)*. [S.l.: s.n.], 2017. p. 268–273.

40 MUKHIYA, Suresh Kumar; RABBI, Fazle; PUN, Ka I; LAMO, Yngve. An architectural design for self-reporting e-health systems. In: *2019 IEEE/ACM 1st International Workshop on Software Engineering for Healthcare (SEH)*. [S.l.: s.n.], 2019. p. 1–8.

41 KUMAR, Mahender; CHAND, Satish. Medhypchain: A patient-centered interoperability hyperledger-based medical healthcare system: Regulation in covid-19 pandemic. *Journal of Network and Computer Applications*, v. 179, p. 102975, 2021. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804521000023>.

42 AYAZ, Muhammad; PASHA, Muhammad F; ALZAHRANI, Mohammed Y; BUDIARTO, Rahmat; STIAWAN, Deris. The fast health interoperability resources (fhir) standard: Systematic literature review of implementations, applications, challenges and opportunities. *JMIR Med Inform*, v. 9, n. 7, p. e21929, Jul 2021. ISSN 2291-9694. Disponível em: <https://medinform.jmir.org/2021/7/e21929>.

43 MAVROGIORGOU, Argyro; KLEFTAKIS, Spyridon; MAVROGIORGOS, Konstantinos; ZAFEIROPOULOS, Nikolaos; MENYCHTAS, Andreas; KIOURTIS, Athanasios; MAGLOGIANNIS, Ilias; KYRIAZIS, Dimosthenis. behealthier: A microservices platform for analyzing and exploiting healthcare data. In: *2021 IEEE 34th International Symposium on Computer-Based Medical Systems (CBMS)*. [S.l.: s.n.], 2021. p. 283–288.

44 NUNES, Jacyana Suassuna; SAMPAIO, Silvio Costa; MALAQUIAS, Ramon Santos; FILHO, Itamir de Morais Barroca. Deploying the observability of the sigsaude system using service mesh. In: *2020 20th International Conference on Computational Science and Its Applications (ICCSA)*. [S.l.: s.n.], 2020. p. 9–15.

45 GOOGLE. *HL7V2; API cloud healthcare; google cloud*. Google, 2022. Acessado em: 08/11/2022. Disponível em: <https://cloud.google.com/healthcare-api/docs/concepts/hl7v2?hl=pt-br>.

46 THUN, Sylvia. The use of fhir in digital health – a review of the scientific literature. *Studies in health technology and informatics*, v. 267, 09 2019.

47 ATLASSIAN. *HL7 v2 data example*. 2022. Acessado em: 13/09/2022. Disponível em: <https://confluence.hl7.org/download/attachments/49644116/VXU_V04\%20-\%202.txt?api=v2>.

48 LUCAS-DOMINGUEZ, Rut; ALONSO-ARROYO, Adolfo; VIDAL-INFER, Antonio; ALEIXANDRE-BENAVENT, Rafael. The sharing of research data facing the covid-19 pandemic. *Scientometrics*, v. 126, n. 6, p. 4975–4990, 2021.