UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

ELISEU GERMANO DA SILVA

Plataforma para Orquestração de Serviços para Cuidados Continuados de Saúde

Goiânia
2019

sistema de bibliotecas ufg

PRPG
PRÓ-REITORIA DE PÓS-GRADUAÇÃO

UFG

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do material bibliográfico: **[x] Dissertação** **[ ] Tese**

2. Identificação da Tese ou Dissertação:

Nome completo do autor: Eliseu Germano da Silva

Título do trabalho: Plataforma para Orquestração de Serviços para Cuidados Continuados de Saúde

3. Informações de acesso ao documento:

Concorda com a liberação total do documento [x] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF da tese ou dissertação.

Eliseu Germano da Silva
Assinatura do(a) autor(a)

Ciente e de acordo:

Assinatura do(a) orientador(a)

Data: 03 / 09 / 2019

[1]Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.

Casos de embargo:
- Solicitação de registro de patente
- Submissão de artigo em revista científica
- Publicação como capítulo de livro
- Publicação da dissertação/tese em livro

ELISEU GERMANO DA SILVA

Plataforma para Orquestração de Serviços para Cuidados Continuados de Saúde

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Sérgio Teixeira de Carvalho

Coorientador: Prof. Dr. Bruno Oliveira Silvestre

Goiânia
2019

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Germano da Silva, Eliseu
Plataforma para Orquestração de Serviços para Cuidados Continuados de Saúde [manuscrito] / Eliseu Germano da Silva. - 2019.
LXXVII, 77 f.: il.

Orientador: Prof. Dr. Sérgio Teixeira de Carvalho; co-orientador Dr. Bruno Oliveira Silvestre.
Dissertação (Mestrado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2019.
Bibliografia. Apêndice.
Inclui tabelas, lista de figuras, lista de tabelas.

1. Orquestração de Serviços. 2. Workflows. 3. Planos de Cuidados. 4. Computação Ubíqua. I. Teixeira de Carvalho, Sérgio, orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

ATA DE DEFESA DE DISSERTAÇÃO

Ata nº 08/2019 da sessão de Defesa de Dissertação de **Eliseu Germano da Silva** que confere o título de Mestre em Ciência da Computação no Programa de Pós-Graduação em Ciência da Computação, na área de concentração em Ciência da Computação.

Aos nove dias do mês de agosto de dois mil e dezenove, a partir das quinze horas, na sala 151 do Instituto de Informática, realizou-se a sessão pública de Defesa de Dissertação intitulada "**Plataforma para Orquestração de Serviços para Cuidados Continuados de Saúde**". Os trabalhos foram instalados pelo Orientador, Professor Doutor Sérgio Teixeira de Carvalho (INF/UFG) com a participação dos demais membros da Banca Examinadora: Professor Doutor Bruno Oliveira Silvestre (INF/UFG - coorientador), membro titular externo; Professor Doutor Alexandre Sztajnberg (IME/UERJ), membro titular externo à instituição; e Professor Doutor Fábio Moreira Costa (INF/UFG), membro titular interno. A Banca Examinadora reuniu-se em sessão secreta a fim de concluir o julgamento da Dissertação tendo sido o candidato **aprovado** pelos seus membros. Proclamados os resultados pelo Professor Doutor Sérgio Teixeira de Carvalho, Presidente da Banca Examinadora, foram encerrados os trabalhos e, para constar, lavrou-se a presente ata que é assinada pelos Membros da Banca Examinadora, aos nove dias do mês de agosto de dois mil e dezenove.

TÍTULO SUGERIDO PELA BANCA

Documento assinado eletronicamente por **Sérgio Teixeira De Carvalho**, **Diretor**, em 09/08/2019, às 17:49, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

http 3bf778e2f7

Documento assinado eletronicamente por **Alexandre Sztajnberg**, **Usuário Externo**, em 09/08/2019, às 17:50, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Fábio Moreira Costa**, **Professor do Magistério Superior**, em 09/08/2019, às 17:50, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Bruno Oliveira Silvestre**, **Professor do Magistério Superior**, em 09/08/2019, às 17:50, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0754522** e o código CRC **3EAE3EE7**.

Eliseu Germano da Silva

Bacharel em Ciência da Computação pelo Instituto de Informática (INF) da Universidade Federal de Goiás (UFG). Durante a graduação em Ciência da Computação foi monitor no INF das disciplinas de Linguagens de Programação e Sistemas Distribuídos. No INF também fez parte de um projeto de pesquisa intitulado Aplicação de Técnicas de Computação Ubíqua, Arquitetura de Software e Inteligência Computacional no Contexto do Monitoramento de Pacientes Domiciliares. Na Embresa Brasileira de Pesquisa Agropecuária (EMBRAPA) trabalhou em um projeto de pesquisa com foco na automatização de processos de genotipagem e gestão de marcadores moleculares. Durante o mestrado no INF/UFG recebeu bolsa da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

Agradecimentos

Ao meu orientador Sérgio Teixeira de Carvalho pelos ensinamentos, disponibilidade e dedicação cedidos durante todo o período de orientação. Sou muito grato ao Sérgio pelos momentos de incentivo à pesquisa, pelo apoio em diversas situações no ambiente acadêmico e pelas várias conversas que tivemos nesses últimos anos. Dar continuidade em alguns de seus trabalhos foi algo muito gratificante.

Ao meu coorientador Bruno Oliveira Silvestre pelo tempo e dedicação cedidos durante o desenvolvimento desse trabalho. A participação do Bruno contribuiu significativamente no andamento desta pesquisa.

À minha família e meus amigos pelo apoio, incentivo, paciência e compreensão nos momentos que precisei me dedicar ao trabalho.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro da pesquisa.

Resumo

Germano, Eliseu. **Plataforma para Orquestração de Serviços para Cuidados Continuados de Saúde**. Goiânia, 2019. 77p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Um plano de cuidados consiste em um conjunto de informações que relaciona pacientes a seus respectivos tratamentos de saúde. Este trabalho propõe a modelagem de um plano de cuidados ubíquo como um conjunto de recursos computacionais centrados no paciente. Esses recursos são modelados como serviços que são orquestrados a partir de *workflows*. Com o objetivo de permitir que um profissional de saúde tenha condições de construir esses *workflows* utilizando serviços que lidem com as especificidades do tratamento de um paciente, apresentamos uma plataforma que recebe a descrição dos *workflows* realizada com o uso de uma Linguagem Específica de Domínio (DSL), orquestra os serviços especificados no *script* da DSL e disponibiliza os *workflows* como novos serviços. Como resultado dessa pesquisa, foram implementados os componentes dessa plataforma, tais como, uma DSL, um modelo computacional para representar os *workflows* em tempo de execução e os mecanismos para viabilizar a comunicação com os serviços. A plataforma foi testada por meio da modelagem de cenários de saúde em que situações práticas de gerenciamento de tarefas de planos de cuidados foram automatizadas por meio da implementação e da composição de serviços *web*. A partir da DSL foram construídos alguns *workflows* para utilizar os serviços registrados na plataforma e avaliar algumas questões a respeito da composição dos serviços, reutilização dos *workflows* e o consumo de recursos computacionais.

Palavras–chave

Orquestração de Serviços, *Workflows*, Planos de Cuidados, Computação Ubíqua.

Abstract

Germano, Eliseu. **Services Orchestration Platform for Continuing Health Care**. Goiânia, 2019. 77p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

A care plan is a set of information that links patients to their therapy. This work proposes modeling ubiquitous care plans as a set of patient-centered computing resources. These resources are modeled as services that are orchestrated by workflows. In order to allow a health professional to be able to build these workflows using services that handle the specificities of a patient's treatment, we propose a platform that receives the description of the workflows, created using a DSL script, orchestrates the services specified in the script and makes workflows available as a new service. Partly as a result of this research, we implemented some of the components of this platform, such as a DSL, a computer model to represent the workflow at runtime, and the required mechanisms for communication with the services. This platform has been tested with the use of healthcare scenarios where practical management situations for care plan tasks have been automated by means of the implementation and registration of web services. Using the DSL, some workflows were built to test the services registered on the platform and to evaluate some issues related to service composition, workflow reuse and computational resource consumption.

Keywords

Service Orchestration, Workflows, Care Plan, Ubiquitous Computing.

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

Um dos grandes desafios investigados em pesquisas nas áreas de ciências da saúde está relacionado à necessidade de realizar o acompanhamento contínuo de pacientes com doenças crônicas [44]. Isso se deve a uma série de fatores que são intrínsecos ao problema de acompanhar continuamente e por tempo prolongado diversos aspectos da saúde humana. Esse problema tem impulsionado o desenvolvimento de pesquisas envolvendo o uso da computação ubíqua em ambientes de assistência domiciliar à saúde, onde por meio da adição de elementos computacionais não invasivos, têm sido investigadas formas de automatizar processos relacionados ao monitoramento de pacientes [17, 19, 43, 54].

A assistência domiciliar à saúde pode ser entendida como um conjunto de serviços médicos e terapêuticos prestados na residência de um paciente para promover, manter ou restaurar a sua saúde [9]. Algumas das principais motivações em torno do uso de elementos computacionais em ambientes de assistência domiciliar à saúde estão relacionadas à possibilidade de melhorar o engajamento das pessoas envolvidas em um tratamento e aumentar a adesão dos pacientes quanto aos procedimentos necessários nesse tratamento [22]. Nessa perspectiva, há propostas de sistemas de monitoramento remoto de pacientes que têm como principal objetivo automatizar alguns desses procedimentos, visando integrar aos ambientes de monitoramento elementos sensoriais (e.g., sensores fisiológicos) e componentes de *software* (e.g., notificadores de eventos) [12, 41].

Em se tratando de acompanhar os cuidados individuais de pacientes com doenças crônicas, uma ferramenta da enfermagem chamada *plano de cuidados* se mostra como uma forte candidata para o uso de tecnologias não intrusivas, e ainda como uma forma de melhorar a adesão desses pacientes aos seus tratamentos. Isso se deve ao fato de que grande parte das atividades relacionadas a um determinado tratamento precisa ser registrada no plano, como os dados fisiológicos coletados do paciente, as tomadas de decisões ao analisar esses dados, e uma série de prescrições que podem variar de acordo com as especificidades da doença tratada. Nesse sentido, é possível pensar em um cenário computacionalmente ubíquo em que dados de um paciente são coletados, enviados e processados em serviços que executam em um ambiente de computação em nuvem, e notificações são enviadas para uma equipe de saúde, especialmente quando há algo de

errado com o estado de saúde do paciente.

Neste trabalho, é apresentada uma plataforma desenvolvida para viabilizar a construção de planos de cuidados como um conjunto de tarefas implementadas na forma de serviços computacionais e que são coordenadas a partir de um modelo baseado em *workflows*. Essas tarefas podem representar, por exemplo, procedimentos de prescrições que um paciente deve seguir durante o seu tratamento, como verificar o horário que uma medicação deve ser ingerida pelo paciente e notificá-lo com mensagens que contenham a posologia do medicamento.

A abordagem utilizada no trabalho é apresentada por meio de uma Linguagem Específica de Domínio (*Domain-Specific Language* - DSL) [20, 25] usada para descrever a sequência em que as tarefas são executadas, com um mecanismo de orquestração dos serviços baseado em um modelo de Redes de Petri. Diante do uso da DSL e dos serviços que são registrados dinamicamente na plataforma, é possível a construção de planos de cuidados adaptáveis e personalizáveis por meio da escolha e do encadeamento dos serviços de acordo com as tarefas específicas de cada tratamento.

1.1 Objetivos

O objetivo geral do trabalho consiste em desenvolver uma solução computacional para viabilizar a construção de planos de cuidados como um conjunto de tarefas implementadas na forma de serviços *web* e que são coordenadas usando um modelo baseado em *workflows*.

O objetivo geral foi subdividido nos seguintes objetivos específicos:

- Definir um modelo computacional para representação de *workflows* compostos por serviços de saúde;
- Desenvolver uma DSL para elaboração dos *workflows*;
- Construir uma plataforma para gerenciar um conjunto de serviços de saúde e invocá-los por meio de orquestrações descritas usando a DSL;
- Validar o trabalho considerando o uso dos artefatos computacionais produzidos em cenários em que tarefas de um plano de cuidados podem ser modeladas a partir de *workflows*;
- Avaliar o trabalho em termos do consumo de recursos computacionais pela plataforma.

1.2 Método de Pesquisa

O desenvolvimento deste trabalho seguiu o método de pesquisa apresentado no diagrama de atividades da UML (*Unified Modeling Language*) mostrado na Figura 1.1. Nesse diagrama estão as principais atividades realizadas e seu agrupamento conforme as etapas seguidas nesta pesquisa.

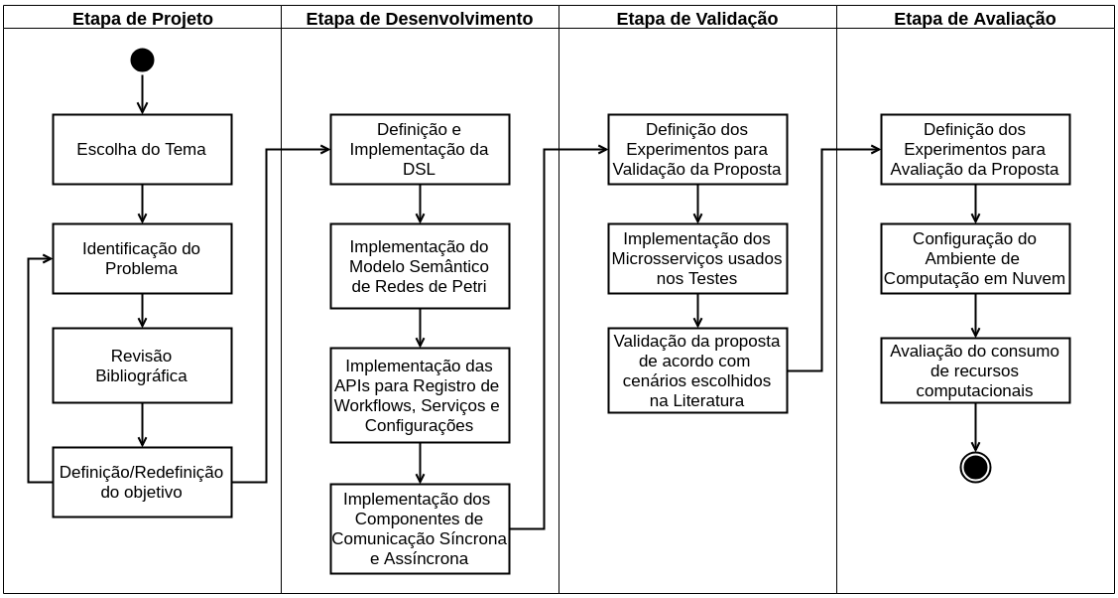

Figura 1.1: *Método de Pesquisa utilizado neste trabalho.*

Assim, as principais etapas do método de pesquisa foram:

- Etapa de Projeto – constituída de um conjunto de atividades onde foram identificados os problemas de pesquisa em torno da construção de sistemas computacionais para gerenciar processos de sistematização de planos de cuidados. Dessa forma, foi proposta uma abordagem para elaboração de um ambiente computacional com foco na composição e coordenação de serviços de saúde. Essa abordagem se apoiou em trabalhos como [8] e [36], que utilizam uma abordagem baseada em *workflows* para cuidados continuados de saúde. Assim, foi proposta a construção de uma plataforma para orquestração de serviços tendo como base uma DSL para compor serviços especializados de saúde a partir de serviços mais básicos.
- Etapa de Desenvolvimento – composta por atividades relacionadas ao desenvolvimento dos artefatos computacionais definidos na etapa de projeto. Nesta etapa os principais componentes da plataforma foram implementados e testados. Esses componentes, assim como a forma na qual eles comunicam entre si, são organizados por meio de um modelo arquitetural em que *workflows* descritos com o uso da DSL orquestram as chamadas aos serviços.

- Etapa de Validação – realizada por meio do uso de componentes implementados na plataforma com o objetivo de conceber e executar *workflows* que representam a modelagem de problemas encontrados na literatura de ciências da saúde. Um dos cenários escolhidos representa uma situação em que a pressão arterial de um paciente precisa ser monitorada continuamente para realização de um diagnóstico de hipertensão. Diante disso, foi construído um *workflow* que orquestra um conjunto de *end-points* de microsserviços que simulam a coleta de dados advindos de sensores fisiológicos do paciente, realiza o processamento desses dados e notifica os interessados.
- Etapa de Avaliação – foi analisado o comportamento da plataforma com respeito ao consumo de recursos computacionais. O objetivo dessa avaliação consiste em identificar tanto a variação do consumo de recursos com o aumento da quantidade de *workflows* simultaneamente executados, quanto a partir da representação de diferentes configurações de *workflows* (e.g., sequenciais, com ramificações e com *subworkflows*).

1.3 Contribuições

As principais contribuições deste trabalho são:

- Uma plataforma para gerenciamento dos *workflows* compostos por microsserviços que implementam tarefas de planos de cuidados;
- Uma DSL para modelagem de *workflows* que representam o encadeamento de tarefas de planos de cuidados;
- Um modelo computacional baseado em Redes de Petri para representação da semântica da DSL;
- A apresentação de uma concepção de modelagem de tarefas de planos de cuidados de enfermagem em casos de cuidados continuados de saúde por meio de serviços computacionais distribuídos.

A ideia inicial de desenvolver um sistema distribuído para disponibilizar recursos computacionais que implementam funcionalidades de planos de cuidados para acompanhar pacientes em ambientes domiciliares, foi publicada no artigo [22], apresentado no *XV Congresso Brasileiro de Informática em Saúde (CBIS – 2016)*[1]. A proposta de desenvolvimento da plataforma para orquestração de serviços de saúde, contendo a arquitetura e uma descrição dos seus principais componentes foi publicada no artigo [23], apresentado no *9th Euro American Conference on Telematics and Information Systems (EATIS –*

[1] http://sbis.org.br/cbis2016

2018)[2]. Por fim, a forma em que a plataforma foi implantada em uma infraestrutura de computação em nuvem, assim como os mecanismos utilizados para que os serviços registrados sejam orquestrados por meio da execução dos *workflows* é descrita no artigo [24], apresentado no *11º Simpósio Brasileiro de Computação Ubíqua e Pervasiva (SBCUP – 2019)*[3].

1.4 Organização

Este trabalho está organizado em quatro capítulos, além deste introdutório. No Capítulo 2 são apresentadas a contextualização e as motivações em torno do trabalho. No Capítulo 3 são apresentados vários aspectos da plataforma de gerenciamento de *workflows* de serviços de saúde, como, a arquitetura, os componentes e os detalhes de como ela foi implementada. No Capítulo 4 é apresentada a forma em que foram realizadas a validação e avaliação da plataforma. No Capítulo 5 é apresentada uma discussão sobre outros mecanismos de composição de serviços, comparando-os com o que é proposto neste trabalho, e, por fim, no Capítulo 6 são descritas as considerações finais desta pesquisa.

[2] http://eatis.org/eatis2018/
[3] http://csbc2019.sbc.org.br/eventos/11sbcup/

CAPÍTULO 2

Fundamentação Teórica

Este capítulo apresenta alguns dos fundamentos teóricos necessários para compreensão do trabalho como um todo. Inicialmente, na Seção 2.1 são apresentados os conceitos relacionados ao uso de planos de cuidados no contexto de assistência domiciliar à saúde. Em seguida, na Seção 2.2 é apresentada uma motivação para o uso de planos de cuidados em ambientes computacionalmente ubíquos. Por fim, na Seção 2.3 são descritos os conceitos utilizados neste trabalho para modelagem de planos de cuidados como um fluxo de tarefas encadeadas, notadamente o conceito de *workflow* e o formalismo de Redes de Petri.

2.1 Sistematização de Planos de Cuidados

Como uma alternativa aos tratamentos hospitalares convencionais, principalmente no que se refere aos tratamentos de longo prazo, a Assistência Domiciliar à Saúde pode ser vista como um conjunto de serviços médicos e terapêuticos prestados em um ambiente residencial visando promover, manter ou restaurar a saúde do paciente [9]. Geralmente esses serviços podem apresentar algumas vantagens em relação aos tratamentos hospitalares, como redução de gastos dos pacientes com internações e minimização de riscos hospitalares [2].

Por outro lado, existem alguns desafios no que se refere à adesão dos pacientes. De acordo com a Organização Mundial da Saúde (OMS), a adesão de um paciente é definida como o grau de comportamento dele em relação às recomendações elaboradas por um prestador de serviços de saúde [44]. Dessa forma, algumas questões relacionadas à autonomia do paciente podem influenciar no tratamento, como, esquecimentos em relação ao cumprimento de tarefas prescritas, falta de conhecimento e habilidade para lidar com os sintomas da doença, má compreensão das instruções prescritas, entre outras. Essas questões podem ser tratadas por meio da sistematização de um plano de cuidados.

Um plano de cuidados é uma ferramenta utilizada na enfermagem, cujo objetivo é permitir aos cuidadores o gerenciamento sistematizado das informações relacionadas aos cuidados prestados aos seus pacientes [31]. Essa ideia tem sido estendida no sentido

de que algumas ações dessa ferramenta podem ser automatizadas, tanto para auxiliar os profissionais de saúde no gerenciamento das atividades do plano quanto para envolver o paciente nos processos relacionados ao seu tratamento. A adição de elementos computacionais autonômicos não invasivos tem sido uma das formas investigadas para automatizar processos dessa ferramenta de saúde [22].

2.2 Planos de Cuidados em Ambientes Ubíquos

Definida em 1991 por Mark Weiser como a onipresença da computação no cotidiano das pessoas a partir da miniaturização dos dispositivos computacionais [55], a computação ubíqua vem se concretizando a partir do advento de novos paradigmas na computação, como a Internet das Coisas e a Web das Coisas [3]. Na área da saúde, uma das principais motivações para se utilizar esse conceito está relacionada à possibilidade do uso não intrusivo de elementos computacionais em determinados ambientes de tratamento [16, 54].

A proposta de ter um Plano de Cuidados Ubíquo leva em consideração cenários em que um conjunto de componentes computacionais integrados ao plano auxiliem tanto o paciente quanto o cuidador na realização das atividades de um tratamento. Para exemplificar essa proposta, na Figura 2.1 é apresentado um diagrama com uma adaptação do modelo proposto por [13]. Trata-se de um modelo de características conhecido como *Feature-Oriented Domain Analysis* (FODA) [34] contendo caraterísticas opcionais, mandatórias e alternativas. Nele pode-se observar algumas entidades de um plano de cuidados (e.g., prescrições, problemas de saúde), sensores (e.g., sensores médicos, sensores de ambiente), componentes relacionados à comunicação em uma rede de computadores (e.g., protocolos HL7 e HTTP) e possíveis elementos autonômicos (e.g., componente de notificação).

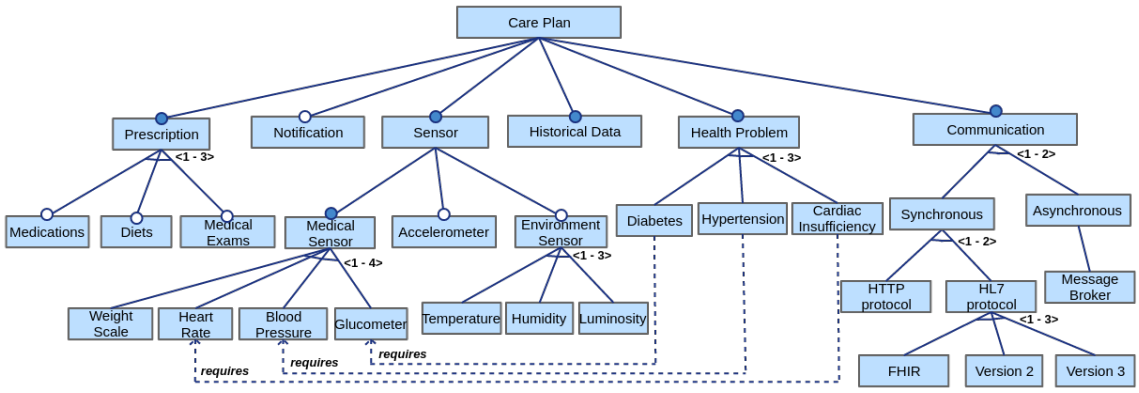

Figura 2.1: *Modelo de características para um Plano de Cuidados Ubíquo (adaptada de [13]).*

O problema do modelo apresentado na Figura 2.1 é sua limitação em relação às características que estão representadas. Esse plano de cuidados não atenderia, por exemplo, um paciente que possua um problema de saúde diferente do que é apresentado (e.g., doença de Alzheimer), que precisa de um outro tipo de prescrição (e.g., atividades físicas) ou que dependa de algum tipo de sensor para o qual, no modelo descrito, não há suporte.

Neste trabalho é proposto um modelo personalizável e orientado a serviços, onde o plano de cuidados é constituído de um conjunto de componentes desenvolvidos de forma independente, mas que podem ser continuamente integrados. Há vários trabalhos que apontam a flexibilidade de realizar a implantação de componentes de software de forma independente como uma das vantagens das arquiteturas de microsserviços [7, 40].

2.3 Modelagem de Planos de Cuidados usando *Workflows*

Nesta seção são apresentados os principais conceitos utilizados neste trabalho em relação à solução proposta para a modelagem de planos de cuidados. Isso é feito por meio da definição de alguns termos utilizados na proposta, como *workflows*, na Subseção 2.3.1, e Redes de Petri, na Subseção 2.3.2. Posteriormente, na Subseção 2.3.3 é apresentada a forma em que Redes de Petri são utilizadas para representar *workflows*. Por fim, na Subseção 2.3.4 é descrita uma forma em que *workflows* podem ser usados para modelar tarefas de um plano de cuidados.

2.3.1 Conceitos de *Workflow*

O termo *workflow* pode ser visto como uma sequência de passos que precisa ser seguida para se realizar uma tarefa. Existem alguns conceitos apresentados por [52] que são relevantes para a compreensão do termo:

- **Tarefas**. Representam os principais elementos que compõem a estrutura de um *workflow*. Essas tarefas podem ser manuais, automáticas ou semi-automáticas. Uma tarefa manual é inteiramente executada por uma ou mais pessoas sem uso de recursos computacionais ou aplicações. Uma tarefa automática é executada sem qualquer intervenção humana, ou seja, quando um recurso computacional executa a tarefa baseando-se inteiramente em dados armazenados. Por fim, uma tarefa semi-automática é aquela em que sua execução depende tanto da intervenção humana quanto de recursos computacionais.
- **Processo**. Indica um conjunto de tarefas que precisam ser executadas com uma possível ordem de execução. Assim como as tarefas, os processos devem ter seu

ciclo de vida limitado. Esse ciclo deve ser formado por construções básicas que podem ser sequenciais, paralelas e seletivas.

- **Execução**. Geralmente a execução de uma tarefa é inicializada por um disparo (*trigger*) a partir de um recurso, um evento externo ou um sinal temporal.

2.3.2 Conceitos de Rede de Petri

Uma Rede de Petri é um tipo de formalismo utilizado para modelagem e análise de processos, sendo composta por *places*, *transitions* e *tokens*. Os *places* são os elementos passivos que representam uma localização geográfica, uma fase ou um estado da rede. As *transitions* são os elementos ativos, cujo objetivo é modelar os processos que são executados durante a passagem de um *token* de um *place* para outro. Os *tokens*, por sua vez, representam objetos que trafegam pelos *places* da rede, sendo os responsáveis por determinar o estado dessa rede em tempo de execução [39, 46]. Na Figura 2.2 é apresentado um exemplo de uma Rede de Petri.

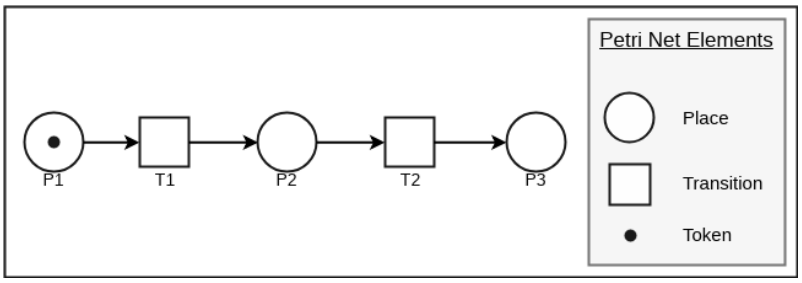

Figura 2.2: *Representação de uma Rede de Petri.*

Como Redes de Petri são utilizadas para modelar situações complexas, seu modelo clássico tem limitações em situações práticas, pois torna-se demasiadamente grande (com muitos *places* e *transitions*) ou inviável para representar os encadeamentos de algumas atividades [52]. Diante disso, esse modelo clássico foi estendido de diversas formas, sendo as principais delas:

- Extensão por cor: utilizada para permitir que dois *tokens* em uma mesma rede sejam distinguidos a partir de um valor (i.e., a cor).
- Extensão por tempo: tem como propósito modelar situações em que é necessário o uso de temporizações durante a execução de uma rede. Dessa forma, os *tokens* recebem um *timestamp* e um valor.
- Extensão hierárquica: tem como objetivo assegurar que seja possível adicionar uma estrutura no modelo, visando criar níveis de abstração que melhorem a legibilidade da Rede de Petri.

Essas extensões, ao serem incorporadas em um modelo clássico, aumentam a capacidade de expressão, o nível de abstração e, consequentemente, facilitam a modelagem

e análise de estruturas com um maior nível de complexidade, tal como é necessário para a construção de *workflows* em um cenário da área de saúde. Em particular, a extensão hirárquica é utilizada na plataforma apresentada neste trabalho para viabilizar a criação de níveis de abstração durante a elaboração dos *workflows*. Os detalhes a respeito de representações de estruturas hierárquicas em Redes de Petri são apresentados na próxima seção.

2.3.3 Mapeamento dos Conceitos de *Workflows* para Redes de Petri

Diante dos conceitos de Redes de Petri descritos na seção anterior, nesta seção são apresentadas algumas construções comuns desse modelo para a representação de *workflows*. Essas construções são utilizadas para definir possíveis caminhos a serem seguidos por um *token* durante o processamento da rede.

Uma das estruturas mais simples de um *workflow*, que pode ser modelada usando uma Rede de Petri, é utilizada para representar um conjunto de tarefas que são realizadas de forma sequencial. Na Figura 2.3 é apresentado um exemplo de Rede de Petri com um caminho sequencial definido a partir de três *places* (representados por P1, P2 e P3) e duas tarefas (representadas pelas transições T1 e T2) que são executadas durante a propagação do *token* na rede, ou seja, durante a transição do *token* de um *place* para outro.

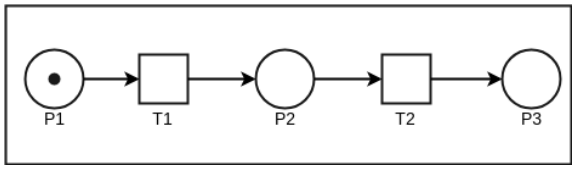

Figura 2.3: *Exemplo de caminho sequencial em uma Rede de Petri.*

Embora estruturas sequenciais sejam amplamente utilizadas para modelagem de processos de um *workflow*, elas não são suficientes para representar todas as tarefas que geralmente são modeladas a partir deles. Dessa forma, existem outras estruturas que permitem a modelagem de tarefas mais complexas, como a modelagem de caminhos paralelos, seletivos e iterativos.

Na Figura 2.4 há um exemplo de um *workflow* que contém uma estrutura que possibilita a modelagem de tarefas que executam em paralelo. Essa figura foi dividida em quatro quadrantes (numerados de 1 a 4) para representar todos os possíveis estados da Rede de Petri dependendo da propagação do *token*. Inicialmente, no quadrante 1, o *token* encontra-se no *place* P1. Ao executar a tarefa T1.1, esse mesmo *token* é propagado para os *places* P2.1 e P2.2, e a rede passa a conter uma estrutura igual a que é apresentada no quadrante 2. Replicações do *token* após a execução de uma tarefa (como no caso da tarefa T1.1) é algo que sempre ocorre quando a estrutura da rede possui um padrão chamado *AND-Split*. Prosseguindo com o processamento, no quadrante 3 é representado o estado

da rede com os *tokens* presentes nos *places* P3.1 e P3.2, enquanto no quadrante 4, há apenas um *token* no *place* P4. Isso significa que ao executar a tarefa T3.1 do *workflow*, os *tokens* foram transformados em um só a partir um padrão conhecido como *AND-Join*.

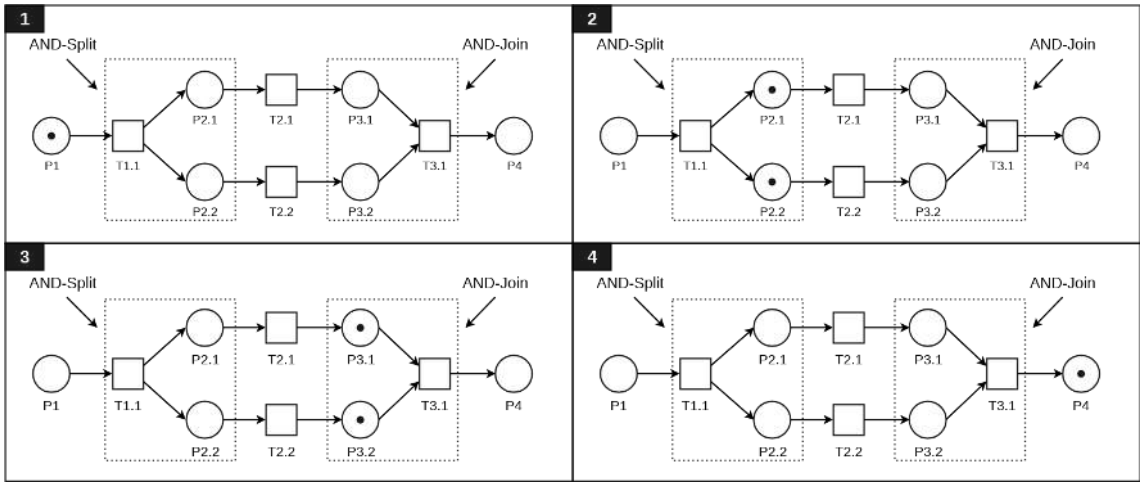

Figura 2.4: *Propagação de um token em uma Rede de Petri com Caminhos Alternativos.*

Há situações em que é necessário representar caminhos alternativos a partir de um *place*. Um exemplo disso é apresentado na Figura 2.5 a partir do uso do padrão *OR-Split*. Nesse exemplo, a propagação do *token* só pode ser seguida a partir de um único caminho, ou seja, a partir do *place* P1 ele deve ir para o *place* P2.1 ou para o *place* P2.2. Isso implica que se a tarefa T1.1 for executada, a tarefa T1.2 é ignorada. Dessa forma, deve haver alguma regra de decisão que viabilize a escolha do caminho a ser seguido.

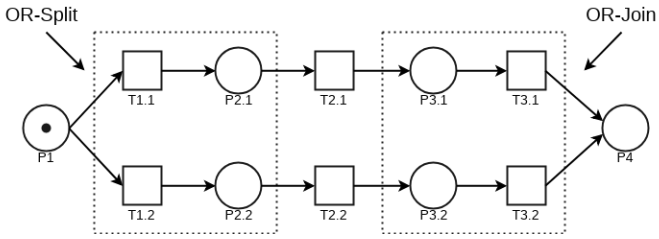

Figura 2.5: *Exemplo de Caminho Seletivo.*

Devido à ampla utilização de Redes de Petri para modelagem de *workflows*, vários padrões relacionados a esse modelo (e.g., *AND-Split*) foram categorizados[1]. A partir do uso desses padrões é possível representar, além dos controles de fluxos básicos, um conjunto de estruturas de controle avançadas, para representar algumas funcionalidades mais elaboradas, como, por exemplo, para modelar pontos de sincronização e de bloqueio em uma rede.

[1] http://www.workflowpatterns.com/

Um outro tipo de estrutura amplamente utiliza em *workflows* é a que representa situações de iteração. Na Figura 2.6 é apresentado um exemplo que ilustra uma situação em que há um caminho iterativo. Nesse exemplo, o *token* representado no *place* P2 pode ser enviado para o *place* P3 em *n* iterações (sendo n $\geq$ 0) antes de ser enviado para o *place* P4.

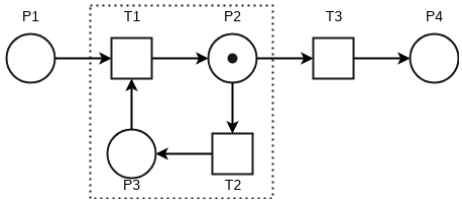

Figura 2.6: *Exemplo de Caminho Iterativo.*

Por fim, uma das notações utilizadas em Redes de Petri de alto nível mais importantes para o desenvolvimento deste trabalho é apresentada na Figura 2.7. Ela contém um exemplo de um *workflow* que utiliza uma estrutura hierárquica representada por um *subworkflow* correspondente à tarefa T1. Isso significa que o *token* localizado no *place* P1 só é enviado ao *place* P4, após ter passado pelo *subworkflow* da tarefa T1. Essa possibilidade de encapsular *subworkflows* em tarefas de um *workflow* viabiliza a representação de múltiplos níveis de abstração em relação à execução de um processo/atividade. Essa possibilidade de analisar um mesmo processo sob diferentes perspectivas é algo discutido na próxima subseção e no Capítulo 3 deste trabalho.

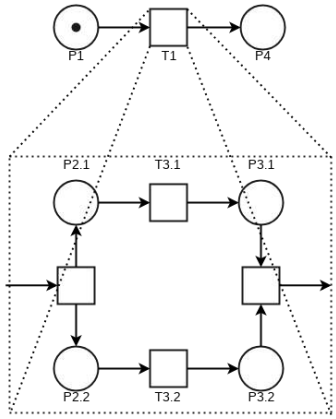

Figura 2.7: *Exemplo do uso de Hierarquias em uma Rede de Petri.*

2.3.4 Planos de Cuidados e a Modelagem de *Subworkflows*

Visando lidar com características de dinamicidade, a abordagem deste trabalho apoia-se na proposta de [8] que considera que um fluxo de tratamento de um paciente pode ser representado por um *workflow*, onde cada uma das etapas desse *workflow* possui um *subworkflow* que pode ser visto como um *workflow* com um nível de especificidade

maior, como pode ser visualizado na Figura 2.8. Assim, considerando que em diferentes situações, diferentes *subworkflows* podem ser utilizados em uma mesma etapa de um *workflow*, a característica de dinamicidade do plano de cuidados pode ser obtida.

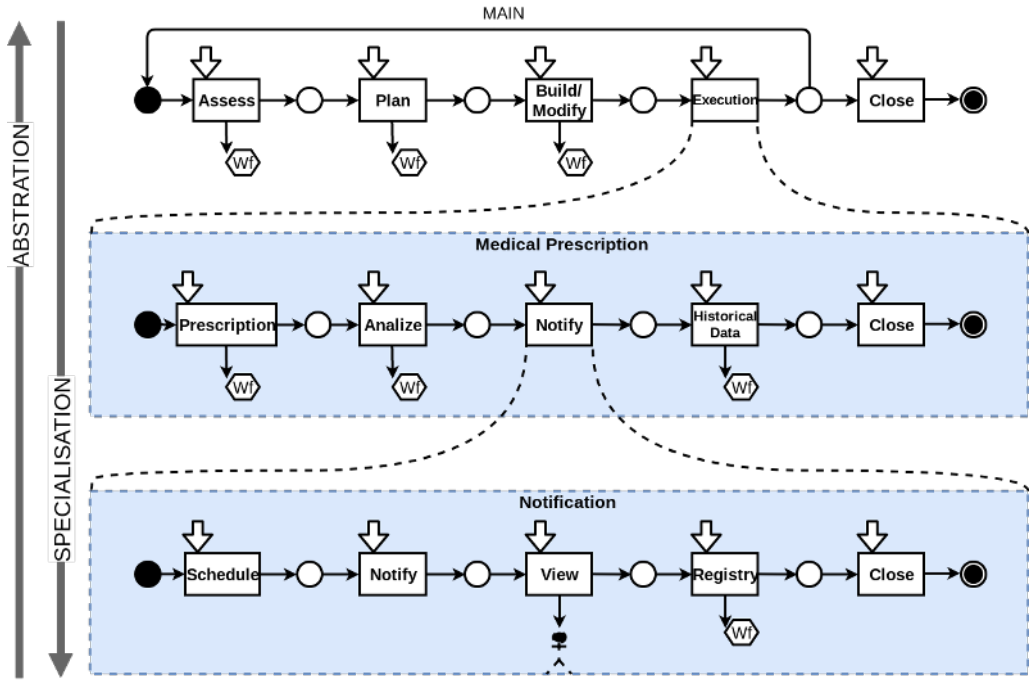

Figura 2.8: *Modelo de subworkflows (adaptada de [8]).*

Diante dessas considerações, no próximo capítulo é apresentada uma forma de viabilizar a modelagem de *workflows* para planos de cuidados como serviços computacionais distribuídos. Para isso, esses serviços são constituídos de composições de microsserviços (serviços de granularidade mais fina), sendo que esses últimos implementam tarefas que representam etapas de *workflows* de planos de cuidados. Seguindo esse raciocínio, as etapas de um *workflow*, ou seja, as implementações dos microsserviços, também podem ser compostas por chamadas a outros microsserviços, possibilitando assim o uso do conceito de *subworkflows*. Em síntese, a partir dessa abordagem foi possível desenvolver uma solução para encadeamento de tarefas de planos de cuidados utilizando características de orientação a serviços, onde as estruturas de um plano são definidas a partir da escolha de um conjunto de microsserviços que melhor atendem às necessidades específicas de um paciente.

2.4 Considerações Finais

Os principais conceitos abordados neste trabalho foram apresentados neste capítulo. Entre esses conceitos estão os Planos de Cuidados de Enfermagem, os *Workflows* e o modelo de Redes de Petri. As informações apresentadas foram inter-relacionadas com

o objetivo de elucidar os elementos que são utilizados na proposta do trabalho, trazendo referências da literatura que mencionam o uso de *workflows* para modelagem de tarefas de planos de cuidados e discutindo uma das formas de representar computacionalmente esses *workflows* por meio do modelo de Redes de Petri.

CAPÍTULO 3

Workflow Management Platform

Este capítulo apresenta a plataforma que viabiliza a integração e orquestração de serviços de acordo com uma descrição especificada em uma Linguagem Específica de Domínio (DSL). Usando essa DSL um usuário do domínio de saúde pode descrever *workflows*, escolhendo dentre um conjunto de serviços registrados na plataforma, aqueles que são adequados em um dado contexto, e definindo um fluxo principal e fluxos alternativos em que eles são executados. Os detalhes em relação aos procedimentos para viabilizar o uso de *workflows* como serviços em um plano de cuidados são descritos nas seções deste capítulo.

3.1 Arquitetura

A abordagem desenvolvida e apresentada neste trabalho para viabilizar a construção e manipulação de *workflows* para planos de cuidados, consiste no desenvolvimento de uma ferramenta computacional cujo foco está no encapsulamento de mecanismos necessários para operar sobre fluxos de execução de tarefas em cenários de saúde. Com o objetivo de representar fluxos com as características de dinamicidade necessárias em um plano de cuidados, desenvolvemos a plataforma chamada de *Workflow Management Platform (WfMP)*. Um dos componentes dessa plataforma consiste em uma DSL com um conjunto de diretivas que considera que as tarefas automatizadas de um plano de cuidados estejam implementadas na forma de microsserviços e que sejam acessíveis por meio de *endponts* na *web* (e.g., *RESTful APIs*).

Na Figura 3.1 é apresentada uma sequência com os passos de interação entre as diferentes entidades que se comunicam com a plataforma. Esses passos compreendem os procedimentos necessários para o registro de serviços (e.g., realizado por um desenvolvedor de *software*), especificação de *workflows* (e.g., realizado por um profissional de saúde utilizando a DSL) e a utilização desses *workflows* na plataforma por aplicações da área de saúde.

No passo 1 há um arco direcionado dos serviços *web* relacionados ao domínio de saúde para a WfMP, que representa o momento em que os serviços são registrados

na plataforma. Esse procedimento deve ser realizado a partir da API da plataforma, que permite configurar na WfMP as informações de baixo nível de abstração necessárias para viabilizar a comunicação entre a plataforma e os serviços externos. Após o registro dos serviços, no passo 2, um usuário do domínio de saúde, utilizando um conjunto de regras e restrições da DSL, escreve um *script* para representar um *workflow*. Uma vez que o *workflow* é constituído de trocas de mensagens entre serviços, estes já devem ser conhecidos pelo usuário do domínio, podendo ser consultados a partir da plataforma em um passo anterior à escrita do *script*.

No passo 3, ainda seguindo o fluxo apresentado na Figura 3.1, o *script* é submetido à plataforma, que por meio de um analisador deve avaliar sintaticamente cada uma das regras descritas. Uma vez que não haja erros sintáticos, um modelo semântico é construído, sendo que este representa a estrutura do *workflow* em um modelo de objetos. Durante a construção do modelo semântico é verificada a disponibilidade dos serviços (passo 4). Com o modelo preenchido, no passo 5 o *workflow* deve ser publicado na plataforma, podendo ser acessado pelas aplicações como um novo serviço, como mostrado no passo 6.

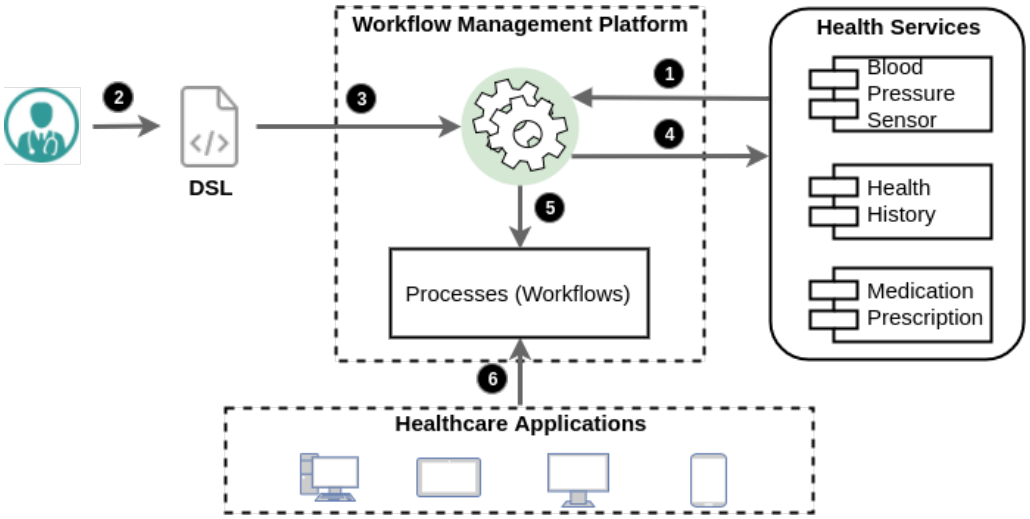

Figura 3.1: *Arquitetura da Workflow Management Platform (WfMP).*

Os passos enumerados têm como objetivo apresentar as principais etapas relacionadas à criação de *workflows*. Essa sequência de interações é fundamental para garantir diferentes níveis de abstração do *workflow* para diferentes tipos de usuários. Assim, um usuário final da plataforma pode ver um *workflow* como um serviço disponibilizado pela API, enquanto que o usuário da DSL o observa como um conjunto de etapas modeladas a partir de eventos, condições e ações. Por fim, um usuário operacional da plataforma, com as tarefas de registrar, configurar e gerenciar os serviços básicos, tem uma visão em um nível de recursos computacionais.

A definição da sequência de procedimentos realizados na plataforma (Figura 3.1) representa não apenas um fluxo de atividades para realizar uma orquestração do serviços de saúde, mas também uma divisão de responsabilidades entre os diferentes tipos de usuários da WfMP. Essa divisão viabiliza a criação de níveis de abstração em relação às representações dos *workflows* que são modelados e executadados na plataforma.

Assim, enquanto um usuário final (e.g., paciente) pode observar um *workflow* como uma atividade composta de um conjunto de procedimentos a serem realizados, um profissional de saúde pode observá-lo/modelá-lo como um conjunto de tarefas compostas pela sistematização de procedimentos clínicos de um plano de cuidados, e, por fim, um profissional de tecnologia da informação pode observá-lo como um encadeamento de chamadas a *end-points* de serviços orquestrados por um elemento central (a WfMP). Esses níveis de abstração são representados na Figura 3.2.

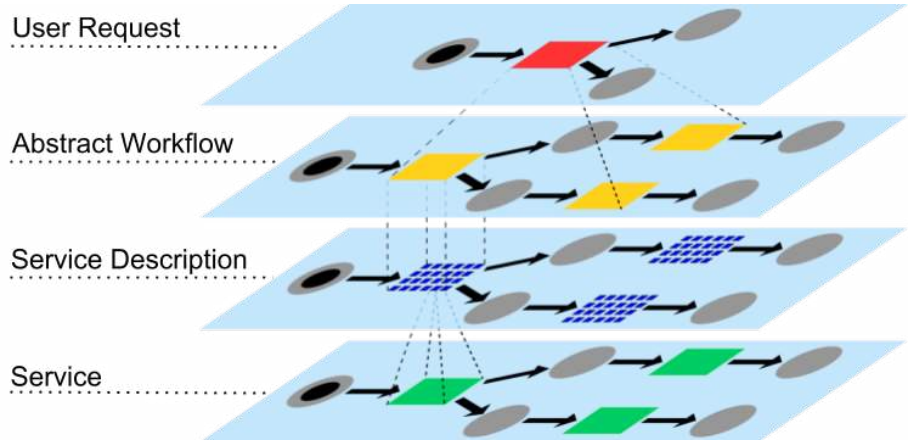

Figura 3.2: *Níveis de abstração de um workflow na WfMP.*

A Figura 3.2 sintetiza as visões que os diferentes tipos de usuários mencionados na arquitetura da WfMP podem ter em relação à modelagem dos *workflows*. Enquanto no nível *User Request* um *workflow* é tido como um processo único, no nível *Abstract Workflow* esses fluxos representam um encadeamento de tarefas, que no contexto da WfMP são as estruturas dos *workflows* relacionados ao domínio de saúde. Essas tarefas, por sua vez, são atividades sistematizadas por meio de composições de serviços computacionais que são representados no nível *Service Description* como um conjunto de recursos registrados na WfMP e que podem ser utilizados na DSL como procedimentos/funções da WfMP. Por fim, no nível *Service* estão os serviços, que são os elementos com nível de abstração mais baixo e que fornecem *interfaces* de acesso (utilizando protocolos de comunicação, localização remota, *interfaces* de entrada e saída de dados, entre outras) aos recursos computacionais externos à plataforma (e.g., *gateways* de sensores fisiológicos de saúde).

3.2 Descrição dos Componentes

A arquitetura interna da WfMP é constituída por uma série de componentes que viabilizam a manipulação de *workflows* desde o processo de análise até o momento em que eles são disponibilizados por meio da API da plataforma. Esses componentes e os fluxos que determinam suas interrelações são apresentados na Figura 3.3. As próximas subseções apresentam as funcionalidades desses componentes.

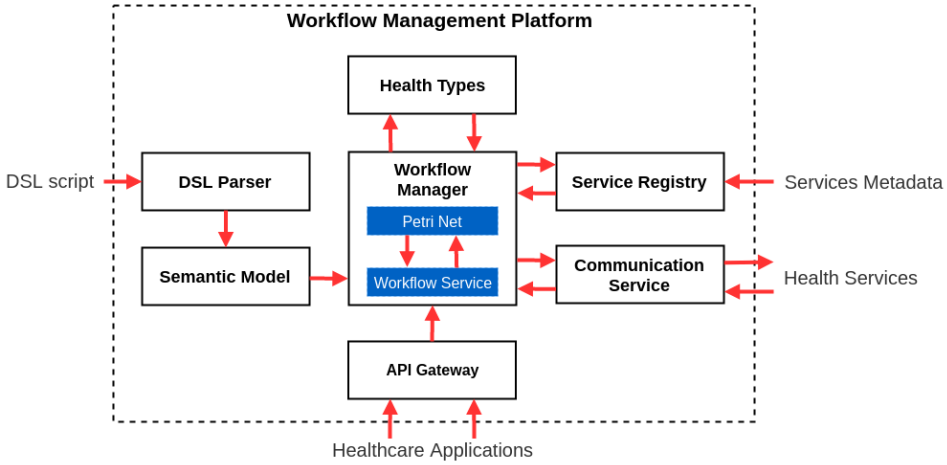

Figura 3.3: *Componentes da WfMP.*

3.2.1 Registro de Serviços

Com o objetivo de garantir a interoperabilidade entre os serviços *web* do domínio de saúde e orquestrá-los por meio do uso de um conjunto de diretivas da DSL, a WfMP oferece o componente *Service Registry* (Figura 3.3) que permite o registro dos serviços externos. Assim, a WfMP oferece por meio da sua API alguns *endpoints* para oferecer os mecanismos para registrar os serviços. Assim, antes que um usuário da plataforma utilize um serviço em um *workflow*, ele deve consultar uma referência desse serviço em um repositório de interfaces. Dessa forma, os mecanismos de comunicação utilizados são feitos de maneira transparente para o usuário da DSL.

Conhecer as interfaces dos serviços também inclui saber quais são os tipos de dados que são passados aos serviços e os tipos que são retornados. Isso permite verificar se os tipos de dados trocados entre serviços são compatíveis. Na WfMP os tipos de dados estruturados válidos devem seguir os modelos representados pelos arquétipos do OpenEHR[1] e/ou tipos que são registrados pelos usuários. O OpenEHR é uma especificação de padrão aberto em informática em saúde que descreve o gerenciamento, armazenamento e troca de

[1] https://www.openehr.org/

dados de saúde em registros eletrônicos de saúde (RES). Essa descrição é feita com o uso de modelos de informação (arquétipos) e de serviços utilizados para representar conceitos clínicos [38, 33].

O componente *Health Types* (Figura 3.3) mantém um repositório com os tipos de dados utilizados pelos serviços registrados na plataforma. Considerando que é comum que a comunicação entre serviços seja feita com o uso de dados estruturados no formato JSON, os tipos registrados pelos usuários na WfMP são representados usando JSON Schema[2]. Dessa forma, durante o processamento e análise de um *workflow*, são verificadas, nos JSON Schemas, as questões de compatibilidade de tipos dos dados trocados entre os serviços. Além disso, o componente *Health Types* mantém um repositório de interfaces contendo os registros de outras informações em relação aos serviços, tais como, a referência remota (localização), os detalhes em relação ao protocolo de comunicação e as propriedades que compõem as mensagens necessárias para realizar essa comunicação (e.g., *headers*).

3.2.2 Definição do Modelo Semântico

O processamento de um *script* da DSL na WfMP é feito em duas etapas, sendo cada uma delas realizada por um componente. No componente *DSL Parser* (Figura 3.3), um *script* passado como entrada é analisado e, caso seja uma entrada válida, uma árvore sintática correspondente é gerada. Em seguida, essa árvore é percorrida pelo componente *Semantic Model* (Figura 3.3) e uma representação do *workflow* em modelo de objetos é criada. Essa abordagem de construir um modelo semântico para uma DSL a partir da árvore sintática gerada por uma entrada válida da linguagem segue a proposta de Martin Fowler para construção de DSLs [20].

O modelo semântico utilizado neste trabalho implementa a estrutura de uma Rede de Petri. Há vários trabalhos que utilizam Redes de Petri para modelagem de *workflows*, como [10] e [29]. Um dos benefícios do uso de Redes de Petri em relação a outros modelos semânticos usados para modelagem de *workflows* (e.g., máquina de estados) é que durante a sua execução é possível que mais de um *place* seja acessado ao mesmo tempo. Essa caraterística facilita a modelagem de fluxos paralelos nos *workflows*. Outro benefício desse modelo, é que já existem esforços em relação à definição de padrões para lidar com fluxo de controle entre várias tarefas (e.g., paralelismo, escolha, sincronização) e para lidar com problemas clássicos como *deadlocks*[3].

[2] http://json-schema.org/
[3] http://www.workflowpatterns.com

3.2.3 Orquestração dos Serviços

Uma vez que *workflows* executados na WfMP são representados usando modelos de Redes de Petri, os mecanismos utilizados para realizar a orquestração dos serviços na plataforma implementam os formalismos desse modelo. O componente da WfMP responsável por realizar a tarefa de orquestração dos serviços em tempo de execução é o *Workflow Manager* (Figura 3.3). De modo geral, esse componente pode ser visto como uma máquina de execução de Redes de Petri cujos *places*, *transitions* e *tokens* contêm dados e procedimentos utilizados para construção e gerenciamento de *workflows*. Enquanto o processo de construção consiste em recuperar em uma base de dados os elementos que compõem a estrutura de um *workflow* e representá-los em um modelo de objetos (mapeamento objeto-relacional), o gerenciamento de *workflows* consiste na coordenação dos fluxos e na manipulação dos dados produzidos em tempo de execução.

A forma na qual o *Workflow Manager* orquestra os serviços é uma consequência de como a orquestração é descrita como um *script* da DSL (Figura 3.3). Isso se deve ao fato de que a estrutura utilizada na sintaxe da DSL foi projetada tendo em vista as características do modelo computacional responsável pelo seu processamento. Nesse sentido, alguns elementos do modelo como os *places* e *transitions* são definidos na forma de procedimentos que seguem uma estrutura do paradigma de programação orientada a eventos conhecida como *event-condition-action* [5].

A gramática da DSL (disponível no Apêndice A) segue as regras na qual as gramáticas são especificadas a partir do gerador de análise sintática ANTLR[4]. A partir das regras apresentadas no Apêndice A é possível analisar como algumas estruturas da linguagem são formadas. Um *workflow* (representado pela palavra reservada *workflow*) é constituído de um identificador e um conjunto de *places*. Os *places*, por sua vez, possuem um identificador e um conjunto de transições, sendo que nessas transições são definidos os eventos, condições e ações. De modo geral, um evento indica a ocorrência de uma determinada situação que precisa ser avaliada (e.g., retorno de um dado processado por um serviço), a condição consiste no processo de avaliação da situação obtida a partir de um evento (e.g., analisar o atributo de um determinado dado), e por fim, a ação representa um conjunto de operações que são executadas quando uma condição assossiada é satisfeita (e.g., chamar um serviço remoto caso o atributo de um determinado dado seja verdadeiro).

A Figura 3.4 apresenta um exemplo de mapeamento entre algumas regras de um *script* escrito usando a DSL para um modelo de Rede de Petri. Na descrição do *workflow* usando a DSL, há um conjunto de procedimentos que estão no escopo de uma tarefa (*task*) chamada *BloodPressureSensor* e sua respectiva transição para a *task HealthcareAnalitics*. Esses procedimentos são representados por um evento (usando a diretiva *case*),

[4] http://www.antlr.org/

uma condição (usando a diretiva *with*) e uma ação (representada pela diretiva *call*). Para determinar a semântica dessas instruções, os eventos, condições e ações são representados nas *transitions* da Rede de Petri. No entanto, apenas quando ocorre uma ação (e.g., *call analyse(BloodPressure)*) um *token* é propagado de um *place* para outro. Portanto, no exemplo apresentado, quando uma nova mensagem *BloodPressure* é recebida (*onReceive*), se (*with*) os valores da pressão sistólica e diastólica estiverem dentro de um determinado intervalo, a ação de realizar uma chamada (*call*) ao *end-point analyze* do microsserviço *HealthcareAnalytics* é realizada. Essa chamada é representada na Rede de Petri pela transição *analyze(BloodPressure)* do *place BloodPressureSensor* para o *place HealthcareAnalitics*.

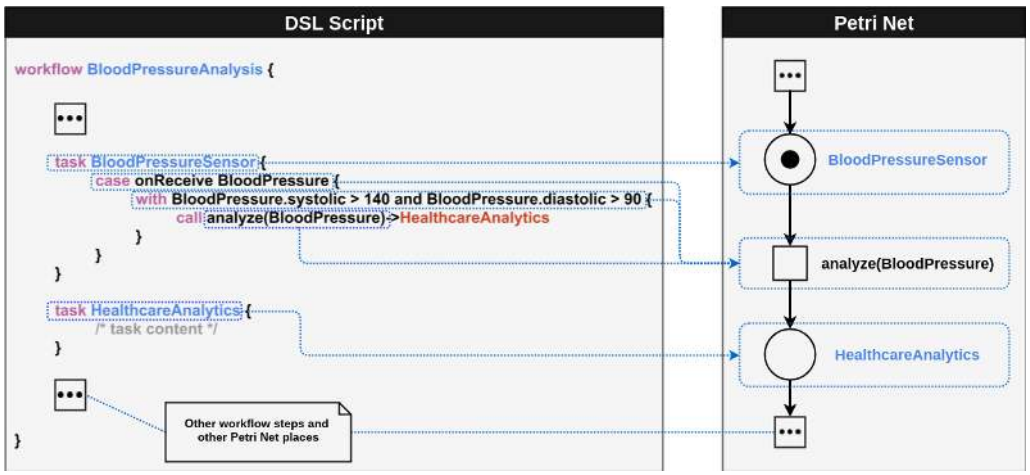

Figura 3.4: *Exemplo de mapeamento das regras de um script da DSL para uma Rede de Petri.*

3.2.4 Mecanismos de comunicação

A WfMP oferece dois mecanismos para viabilizar a comunicação entre o *Workflow Manager* e os serviços de saúde, sendo eles por meio de comunicação síncrona e assíncrona. Para lidar com a comunicação síncrona é utilizado o protocolo HTTP (*Hypertext Transfer Protocol*), enquanto para comunicação assíncrona são utilizados os protocolos AMQP (*Advanced Message Queuing Protocol*) e MQTT (*Message Queuing Telemetry Transport*). Esses mecanismos estão contidos no componente *Communication Service* da plataforma (Figura 3.3).

Para lidar com a comunicação assíncrona, o componente *Communication Service* possui um conjunto de procedimentos necessários para que a troca de dados entre os serviços e o *Workflow Manager* seja feita a partir de filas de mensagens. Dessa forma, é possível que alguns dados sejam enviados dos serviços para a plataforma (e vice-versa) por intermédio de um componente de *software* que permite que haja um desacoplamento temporal entre as entidades comunicantes.

Para lidar com a comunicação síncrona entre a plataforma e os serviços que estão nela registrados, o componente *Communication Service* faz um mapeamento entre a forma que uma chamada a um serviço *web* é especificada como um conjunto de regras da DSL e a maneira como ela é de fato realizada em um nível de rede. Portanto, esse componente é responsável por montar uma mensagem HTTP, adicionando os *headers* necessários e definindo a estrutura de acordo com os métodos de solicitação de cada requisição.

A WfMP oferece ainda um mecanismo de comunicação entre o *Workflow Manager* e as aplicações de saúde chamado *API Gateway*. Esse componente é responsável por exportar os *workflows* gerados na plataforma como um conjunto de serviços especializados. São algumas das funções desse componente realizar a publicação, manutenção, monitoramento e controle de acesso aos *workflows*.

3.3 Implementação dos Componentes

Esta seção tem como objetivo apresentar os detalhes de implementação dos componentes descritos na Seção 3.2. Também são mencionadas as ferramentas computacionais utilizadas, assim como a forma em que elas foram empregadas durante a construção dos componentes de *software* da WfMP.

3.3.1 Componentes de Análise Sintática

A DSL da WfMP tem como objetivo funcionar como um mecanismo de interação entre os especialistas do domínio de saúde e os serviços da plataforma. Sendo assim, ela é um artefato importante da plataforma. No entanto, desde o início do projeto, há a consciência do desafio de se construir uma linguagem para o domínio de saúde que possa refletir o encadeamento de tarefas de um plano de cuidados e que, ao mesmo tempo, tenha uma sintaxe e semântica adequadas ao entendimento de pessoas que não são da área de computação. Logo, discutir as questões relacionadas à usabilidade da DSL pelos profissionais e, possivelmente, desenvolver uma DSL gráfica que seja semanticamente equivalente à DSL textual são pontos importantes que devem ser considerados no escopo do projeto. No entanto, esses pontos não foram o foco principal deste trabalho. Nesse sentido, a versão atual da DSL foi desenvolvida tendo como foco a representação de tarefas por meio de *workflows* e considerando o modelo computacional que representa a semântica de execução de suas construções.

Os componentes de análise sintática da DSL foram desenvolvidos utilizando uma ferramenta chamada ANTLR. A função do ANTLR na implementação da linguagem consiste em receber um *script* construído com o uso das regras da DSL e, seguindo as regras de produção da gramática (Apêndice A), realizar a análise sintática do *script*,

produzindo uma árvore sintática e um conjunto de *interfaces* que viabilizam o acesso a essa árvore de acordo com a linguagem de programação utilizada. O ANTLR gera analisadores para diversas linguagens de programação, como C, C#, Java, JavaScript, Objective-C, Perl, Python, Ruby. A DSL implementada neste trabalho foi desenvolvida usando a linguagem Java[5] na versão 8.

Diante da árvore sintática e das *interfaces* para acessá-la, a árvore é percorrida e, caso não sejam detectados erros de sintaxe e/ou de tipagem de possíveis dados trocados entre os serviços, a Rede de Petri é construída. Uma vez que o modelo semântico é construído, sua representação em um modelo de objetos é mapeada para uma representação relacional de banco de dados e persistida em um Sistema Gerenciador de Banco de Dados PostgreSQL[6]. O *workflow* registrado no banco de dados, pode então ser consultado e executado por meio de invocações realizadas nas APIs da WfMP. Além disso, o *workflow* pode ainda ser utilizado como um *subworkflow* durante as especificações de novos *workflows* usando a DSL.

3.3.2 Componentes do Modelo Semântico

O modelo semântico de uma DSL consiste em um modelo de objetos no qual a representação das entidades que compõem a estrutura da linguagem são expressas em termos de classes do paradigma de orientação a objetos. Enquanto esse modelo tem a função de representar as estruturas semânticas da liguagem, a DSL tem a função de preenchê-lo ao passo de um processo de análise sintática. Essa abordagem permite um nível de desacoplamento entre a sintaxe da linguagem e o modelo que representa sua semântica, viabilizando que ambos possam evoluir separadamente [20].

O diagrama de classes da UML na Figura 3.5 ilustra os principais elementos que compõem o modelo semântico de Redes de Petri utilizado na WfMP. Embora trate-se de uma representação simplificada do modelo real, é possível identificar a forma em que o modelo está estruturado na WfMP.

Como pode ser observado na Figura 3.5, uma Rede de Petri é constituída pelas classes *PetriNet*, *Place* e *Transition*. A classe *Transition* possui uma referência para um *place* de origem (*source*) e um *place* de destino (*target*) para permitir a propagação dos *tokens* durante a execução da rede. Além disso, a classe *Transition* é composta por *Event*, *Condition* e *Action*, que são responsáveis por definir o comportamento de uma transição.

Diante da forma em que as Redes de Petri são estruturadas na WfMP, realizar o seu processamento, ou seja, fazer a propagação dos *tokens* pelos *places* da rede, consiste basicamente em realizar um percurso do *place* inicial até um *place* que não possua

[5] https://docs.oracle.com/en/java/
[6] https://www.postgresql.org/

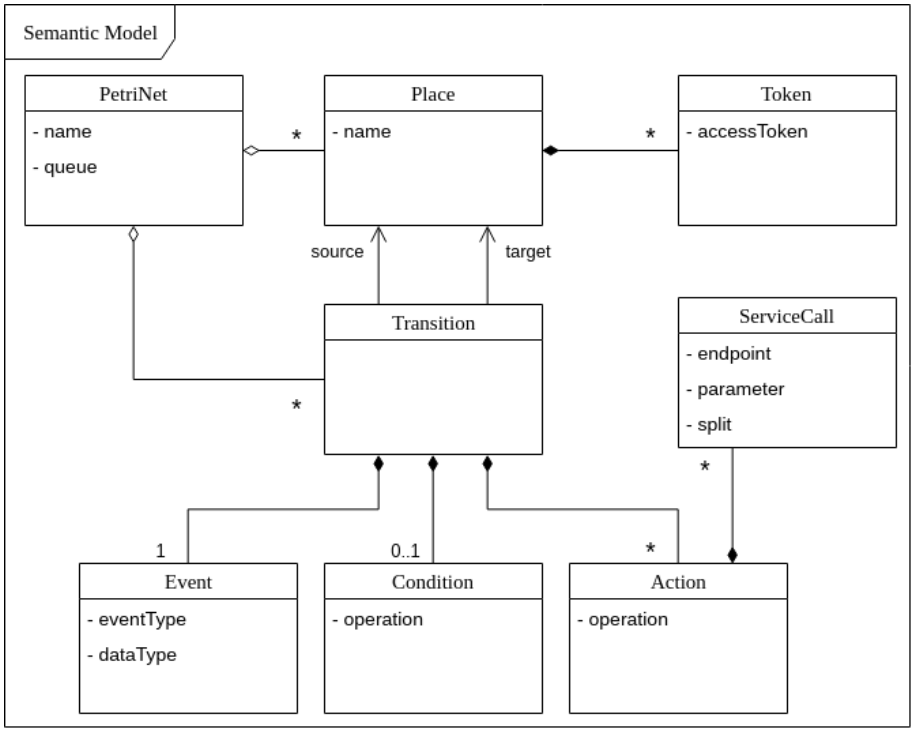

Figura 3.5: *Principais elementos do Modelo Semântico.*

transição. Vale mencionar que a decisão do caminho a ser realizado durante o percurso é algo determinado por elementos que compõem as transições (eventos, condições e ações) e em alguns casos pela ordem de propagação de um conjunto de *tokens* na rede, sendo esse caminho o fator determinante para a realização da orquestração dos serviços.

A implementação do Motor de Execução de Redes de Petri na WfMP foi realizada utilizando o *framework* Spring Boot[7] e um conjunto de módulos do Spring Framework, como Spring Data[8], Spring AMQP[9] e o Spring Security[10].

3.3.3 Componentes para Comunicação Síncrona e Assíncrona

Uma das principais funções da plataforma consiste em realizar a orquestração de serviços. Para isso, há componentes que são utilizados para viabilizar a comunicação, tais como, componentes para comunicação síncrona e assíncrona, componentes para controlar o registro dos serviços, repositórios de tipos de dados de saúde, entre outros. Do ponto de vista do processo de orquestração de serviços na plataforma, o componente de principal importância é o que viabiliza a comunicação através da *web*. Esse componentes é o *Com-*

[7] https://spring.io/projects/spring-boot
[8] https://spring.io/projects/spring-data
[9] https://spring.io/projects/spring-amqp
[10] https://spring.io/projects/spring-security

munication Service, que utiliza para comunicação síncrona e assíncrona, respectivamente, os protocolos HTTP e AMQP/MQTT.

Considerando ainda que os recursos computacionais utilizados em um *workflow* podem fazer parte de uma arquitetura de microsserviços, é comum que a comunicação ocorra por meio de requisições assíncronas. Para lidar com a troca de mensagens assíncronas é utilizado o RabbitMQ[11], um Middleware Orientado por Mensagens (MOM). Uma das principais motivações para se utilizar o RabbitMQ na WfMP consiste em viabilizar o assincronismo na troca de mensagens com um desacoplamento temporal entre as entidades comunicantes. Além disso, algumas características do *middleware* se tornaram partes importantes para a solução apresentada pela WfMP, tais como, os diferentes tipos de roteamento de mensagens entre produtores e consumidores.

Na arquitetura da WfMP é utilizada comunicação baseada em tópicos, ou seja, uma mensagem é enviada de um produtor para um tópico, e então é encaminhada, por meio do uso de uma chave de roteamento, para todas as filas de mensagens que satisfazem esse padrão. Além disso, foi estabelecido que cada *workflow* da plataforma possua uma fila de mensagens para receber dados durante a orquestração dos serviços. A Figura 3.6 apresenta uma representação da forma em que as mensagens são encaminhadas para os *workflows* por meio do RabbitMQ.

De acordo com o fluxo da Figura 3.6, os serviços enviam mensagens contendo metadados com uma chave de roteamento para um tópico chamado *workflow_data*. Quando uma mensagem é recebida pelo *workflow_data*, é verificado se a chave de roteamento dessa mensagem coincide com as chaves de roteamento das filas (*Queues*) vinculadas (*bind*) ao tópico. Dessa forma, receberão essa mensagem todas as filas (e consequentemente todos os *workflows*) que estejam vinculados ao tópico por esse padrão.

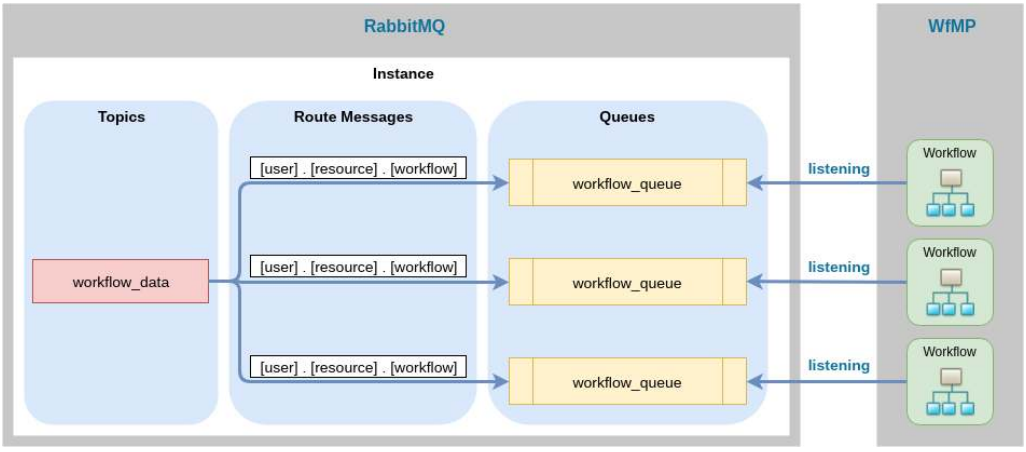

Figura 3.6: *Propagação das mensagens assíncronas.*

[11] https://www.rabbitmq.com/

Essa abordagem baseada em chaves de roteamento permite que a qualquer momento uma nova fila possa receber dados de um determinado tópico, bastando que ela esteja vinculada ao tópico por meio de uma chave de roteamento. Esse mecanismo para encaminhamento de mensagens influencia significativamente na arquitetura do componente de comunicação da WfMP, pois, a qualquer momento, novos *workflows* que sejam registrados precisam receber dados dos serviços externos sem que haja um acoplamento temporal. Dessa forma, os dados podem ser enviados dos serviços para os *workflows* sem que haja qualquer tipo de acoplamento entre eles.

Ainda de acordo com a Figura 3.6, há uma padronização em relação à definição das estruturas das chaves de roteamento de acordo com algumas regras definidas pelo próprio RabbitMQ (*Route Messages*). As chaves de roteamento que vinculam o tópico *workflow_data* às filas de mensagens têm a estrutura [*user*].[*resource*].[*workflow*], onde o elemento [*user*] representa o identificador de um usuário do *workflow* na plataforma, o elemento [*resource*] representa o tipo de recurso registrado na plataforma ao qual a mensagem recebida pertence e, por fim, o elemento [*workflow*] representa o identificador do *workflow* em que a mensagem é destinada. Por exemplo, a chave "*user00245.bloodpressure.workflow00134*" é uma chave válida para a WfMP, desde que *user00245* seja um usuário válido, *bloodpressure* seja um recurso válido e *workflow00134* seja um *workflow* registrado na plataforma.

RabbitMQ define que os elementos de uma chave de roteamento composta devem ser delimitados por "." (ponto). Ele também permite o uso de alguns símbolos para representar padrões de comportamentos para as chaves, como o "*" (asterisco) para que uma chave de roteamento tenha um elemento de uma posição específica (por exemplo, a chave de roteamento de "A.*.*.B.*" corresponde apenas às chaves onde a primeira palavra é "A" e a quarta palavra é "B") e o "#" (cerquilha) para indicar uma correspondência em zero ou mais chaves (por exemplo, uma chave de roteamento "A.B.#" corresponde a qualquer chave iniciada por "A.B"). Dessa forma, a chave "*user00245.#*" é um exemplo de chave válida que permite que uma mensagem recebida seja encaminhada para todas as filas dos *workflows* pertencentes ao usuário *user00245*. Por outro lado, a chave "*user00245.bloodpressure.#*" permite que uma mensagem seja encaminhada apenas para as filas dos *workflows* pertencentes ao usuário *user00245* e que recebem mensagens que pertencem ao recurso *bloodpressure*.

Uma vez que é utilizada uma única fila no RabbitMQ para cada *workflow* registrado na WfMP e considerando que os *workflows* podem receber dados de diferentes serviços utilizados durante um processo de orquestração, essas filas podem receber diferentes tipos de dados. Dessa forma, a WfMP possui um componente chamado *Resource Adapter* que é responsável por adicionar junto aos metadados das mensagens, uma informação que indica o tipo de dado que ela representa. Esses tipos de dados são registrados

durante o processo de registro dos serviços na plataforma e são verificados durante a execução dos *workflows*.

Apesar do metadado contendo o tipo de dado da mensagem possibilitar a verificação da informação extraída de uma fila do RabbitMQ, os *workflows* recebem os dados das filas do RabbitMQ de forma passiva e pode ser que o dado recebido não seja do tipo de dado esperado em um determinado momento do processamento de um *workflow*, mas que possa ser útil em um momento posterior. Dessa forma, foi necessário determinar uma estratégia para viabilizar o recebimento de todos os possíveis tipos de dados esperados de um *workflow* em execução.

A Figura 3.7 apresentada a solução utilizada na WfMP para lidar com o recebimento de mensagens com diferentes tipos de dados no RabbitMQ. A solução consiste no uso de uma fila por *workflow* no RabbitMQ e uma fila por tipo de dado do *workflow* na WfMP. Dessa forma, sempre que uma mensagem é recebida por intermédio de uma fila do RabbitMQ, seu tipo é verificado e, então, o dado é encaminhado para uma fila da WfMP (e.g., DT-01).

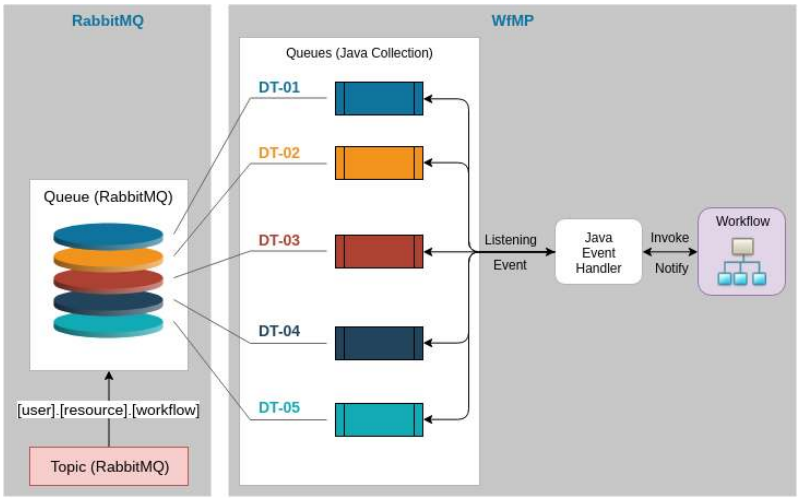

Figura 3.7: *Recebimento das mensagens assíncronas.*

A WfMP mantém uma estrutura *hash* (*Queues* do *Java Collection* na Figura 3.7) em que as chaves são representadas pelos tipos de dados esperados por um *workflow* e os valores são listas que armazenam os valores de *input* recebidos pelo mesmo *workflow*. Dessa forma, enquanto ocorre o processamento de um *workflow* e espera-se o retorno de um dado de um determinado serviço (ou seja, uma entrada do *workflow*), é realizada uma operação de escuta na fila que contém o tipo de dado esperado. O elemento responsável por essa operação é o componente *Java Event Handler*, que por sua vez utiliza uma técnica de sincronização baseada em Monitores de Hoare [30] para notificar um *workflow* quando um dado do tipo esperado é recebido.

Vale mencionar que, com o uso da API da WfMP, é possível registrar um *timeout* para os dados contidos em cada uma das listas de dados de um *workflow*. Isso permite

determinar a validade de um dado de acordo com suas características (e.g., determinar o tempo de validade de um dado que guarda as informações da pressão arterial de um paciente). Assim, um profissional de saúde, por meio de uma aplicação com acesso às funcionalidades da API da WfMP, poderia configurar os tempos de aceitação de cada um dos tipos de dados em cada *workflow* que ele tenha acesso.

3.4 Representação de Serviços de Saúde

A plataforma discutida neste trabalho tem como objetivo coordenar um conjunto de tarefas implementadas na forma de serviços. Assim, *workflows* representam a forma/organização em que as tarefas são executadas de acordo com um determinado fluxo. Para facilitar o entendimento de como *workflows* são executados, esta seção foi dividida em duas partes. Enquanto a Subseção 3.4.1 apresenta a visão de usuário da plataforma em relação à modelagem e execução dos *workflows*, a Subseção 3.4.2 traz uma visão de como esse processamento ocorre do ponto de vista do modelo computacional responsável pela orquestração dos serviços que implementam as tarefas dos *workflows*.

3.4.1 Perspectiva dos Usuários dos *Workflows*

Como forma de analisar a representação de possíveis *workflows* usados na área da saúde na WfMP, foi considerado um plano de cuidados para pacientes de hipertensão (*HypertensionCarePlan*), apresentado na Figura 3.8 a partir do *workflow HypertensionCarePlan* (linha A). A mesma Figura 3.8 apresenta os *subworkflows* (linhas B, C e D): *HypertensionDiagnostics*, *BloodPressureAnalyzer* e *HypertensionTreatment*. Do ponto de vista do domínio de saúde, o objetivo desse cenário consiste em representar um plano de cuidados para hipertensão dividido em duas etapas: etapa de diagnóstico e etapa de tratamento. Sendo assim, na etapa de diagnóstico os dados fisiológicos de um paciente seriam analisados, e, ocorrido algum problema, a etapa de tratamento seria executada.

Devido ao nível de abstração do *workflow HypertensionCarePlan*, não é possível saber os detalhes de como ocorre o processso de diagnóstico e o processo de tratamento da doença. No entanto, nos seus *subworkflows* esses processos são descritos com um nível de especificidade maior. O mesmo comportamento ocorre com o *workflow HypertensionDiagnostics*, que por sua vez é dividido na etapa de análise dos valores de pressão arterial (*BloodPressureAnalyzer*) e em uma etapa de avaliação médica (*MedicalEvaluation*). Nesse caso, a forma de como ocorre a análise dos valores de pressão arterial é definido apenas no *subworkflow BloodPressureAnalyzer*.

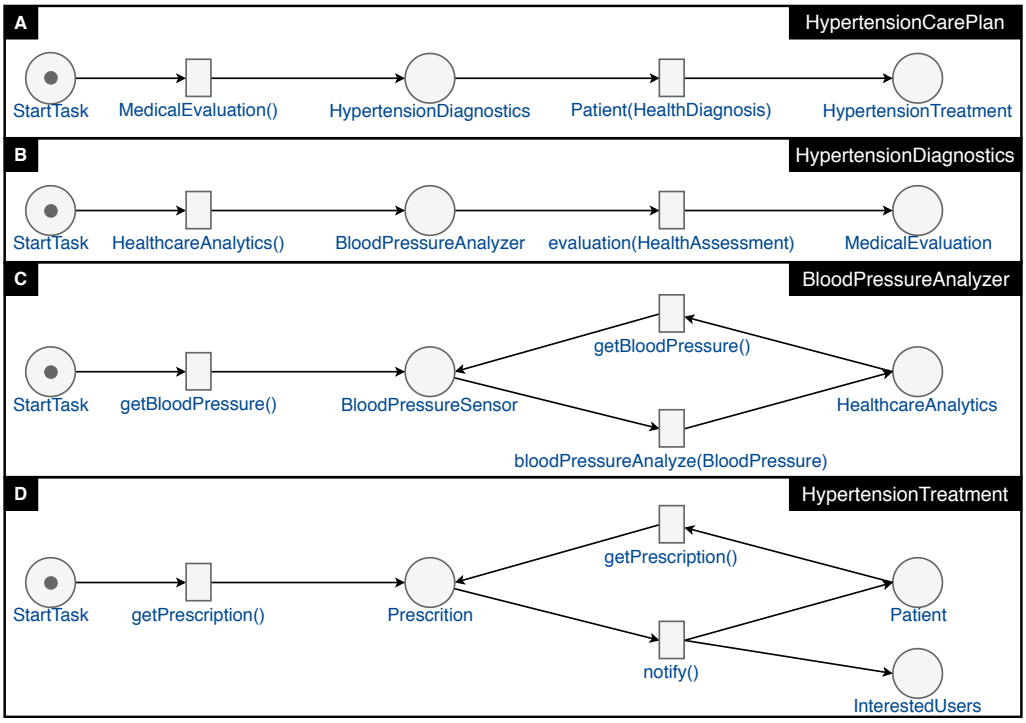

Figura 3.8: *Exemplo de workflow de um Plano de Cuidados para Hipertensão.*

A partir desse exemplo, pode ser observado que ao seguir o modelo proposto por Eric Browne [8], uma etapa de um *workflow* pode ser representada a partir de diferentes *subworkflows* com um nível maior de especificidade. No caso do exemplo apresentado, o *workflow* que realiza uma análise dos valores de pressão arterial segue as diretrizes de um modelo de diagnóstico de hipertensão proposto por [15], que tem como objetivo diferenciar casos de pacientes que possuem *white coat hypertension* (também conhecida como a síndrome do jaleco branco) daqueles que de fato possuem hipertensão. No entanto, caso fosse necessário, um outro modelo poderia ser utilizado, o que possivelmente ocasionaria uma mudança em relação ao *workflow* para análise dos dados de pressão, mas que não influenciaria diretamente nos *workflows* de nível de abstração mais alto, como o *HypertensionDiagnostics* e o *HypertensionCarePlan*.

Considerando que os *workflows* discutidos neste trabalho são constituídos de serviços computacionais e que do ponto de vista da plataforma cada *workflow* registrado ao ser executado também tem o comportamento de um serviço, alguns mecanismos foram implementados para viabilizar a representação de um *workflow* como um serviço. Esses mecanismos influenciaram tanto na forma em que os *workflows* são representados na DSL, quanto na forma em que eles são processados pela plataforma.

A DSL da WfMP tem como objetivo funcionar como um mecanismo de interação entre os especialistas do domínio de saúde e as abstrações dos recursos (e.g., identificadores dos serviços) da plataforma. De modo geral, um *script* da linguagem é constituído

de blocos compostos por elementos que seguem o padrão *Event-Condition-Action*, que tem sido utilizado em diversos trabalhos na área da saúde [32, 37, 56]. Assim, cada etapa de um *workflow* contém pelo menos uma regra do tipo *Event-Condition-Action*. Semanticamente, esse modelo representa situações em que as ações são disparadas por eventos sempre que é satisfeita uma ou mais condições. Essas ações, por sua vez, representam as chamadas aos serviços remotos (externos à plataforma) e ao mesmo tempo uma transição entre as tarefas durante o processamento de um *workflow*.

Para um melhor entendimento da estrutura de um *script* da DSL, na Figura 3.9 é exemplificado o código que traduz a representação gráfica do primeiro *workflow* (*HypertensionCarePlan*), apresentada na Figura 3.8. Esse *workflow* é composto por três tarefas (*tasks*), onde a primeira delas (*start stask*) representa o ponto de partida do *HypertensionCarePlan*, o qual realiza uma chamada ao *endpoint MedicalEvaluation* do serviço *HypertensionDiagnostics*. Uma vez que a chamada (*call*) da *start task* é realizada, o fluxo de controle do *workflow* é direcionado para a tarefa seguinte para receber o retorno do serviço *HypertensionDiagnostics*. Como os tipos de dados de *input* e *output* dos serviços são registrados na plataforma, quando o dado de retorno é recebido pelo *workflow* (*onReceive*), é verificado nos *cases* (com o uso dos operadores *isTrue* e *isFalse*) o valor do atributo do tipo *boolean* chamado *hasProblem*. Caso o primeiro *case* seja satisfeito, é realizada uma chamada ao *endpoint Patient* do serviço *HypertensionTreatment* e o fluxo de controle do *workflow* é direcionado para a tarefa seguinte, a fim de receber o retorno desse serviço. Se por outro lado, o segundo *case* for satisfeito, o *workflow HypertensionCarePlan* é finalizado. Dessa forma, ao receber o dado de retorno (*onReceive*) *Prescription*, se o valor do atributo *endTreatment*, verificado pelo operador da linguagem *currentDateEqual* possuir a data atual, o *workflow HypertensionCarePlan* é finalizado.

3.4.2 Perspectiva de Execução do Modelo Semântico

Na WfMP, os *workflows* são modelados com o uso de uma DSL, seguindo as diretrizes de Martin Fowler [20]. De acordo com essa abordagem, uma DSL é composta por uma sintaxe e um modelo computacional que representa a semântica de um *script* (programa) construído usando essa sintaxe, denominado modelo semântico. O modelo semântico utilizado na WfMP é conhecido como Rede de Petri, escolhido principalmente devido à sua forte aceitação na representação de *workflows*.

O processamento de um *script* da DSL na WfMP é feito em duas etapas. Na primeira o *script* é analisado sintaticamente e, caso não haja erros, uma árvore sintática correspondente é gerada. Em seguida, essa árvore é percorrida e uma representação do *workflow* em modelo de objetos é criada. Essa representação é persistida em um banco de dados e, quando o *workflow* é invocado por meio da API da plataforma, o modelo

```
workflow HypertensionCarePlan {
        start task {
                call MedicalEvaluation()->HypertensionDiagnostics
        }

        task HypertensionDiagnostics {
                case onReceive HealthDiagnosis {
                        with isTrue(HealthDiagnosis.hasProblem) {
                                call Patient(HealthDiagnosis)->HypertensionTreatment
                        }
                }
                case onReceive HealthDiagnosis {
                        with isFalse(HealthDiagnosis.hasProblem) {
                                end.workflow
                        }
                }
        }

        task HypertensionTreatment {
                case onReceive Prescription {
                        with currentDateEqual(Prescription.endTreatment) {
                                end.workflow
                        }
                }
        }
}
```

Figura 3.9: *Exemplo de workflow modelado por meio de um script da DSL da WfMP.*

de objetos é recuperado e executado de acordo com a semântica que ele representa. Em outras palavras, executar um *workflow* na WfMP é equivalente a executar um modelo semântico que representa uma Rede de Petri.

A Figura 3.10 apresenta os principais componentes de *software* envolvidos no processo de execução de *workflows*. Do lado esquerdo, há um servidor contendo componentes que persistem os *workflows* e gerenciam o processo de orquestração (WfMP-Manager). Na parte central, há um conjunto de componentes responsáveis por executar os *workflows* (*workers*) e um componente responsável por realizar a distribuição de carga entre eles (*Load Balancer*). Por fim, do lado direito, há um conjunto de microsserviços externos à plataforma que oferecem recursos que podem ser acessados por meio de *endpoints* disponibilizados por meio de um processo de registro de serviços.

Uma vez que os *workflows* executados na plataforma são modelados como Redes de Petri, os *workers* são máquinas de execução de Redes de Petri que realizam as orquestrações conforme essas redes são percorridas. Sob a perspectiva do que é discutido neste trabalho, uma orquestração é uma composição de serviços que possui um elemento central (*workflow*) para controlar e coordenar a execução de cada chamada a serviços externos da plataforma. Considerando o modelo de Redes de Petri, o processamento de *workflows* nos *workers* utiliza uma semântica em que os *places* representam elementos passivos e as *transitions* os elementos ativos de uma orquestração.

Tomando como exemplo o terceiro *workflow* da Figura 3.8 (chamado *BloodPressureAnalyzer*), a ação de executar um serviço para obter um dado de pressão arterial ocorre na *transition getBloodPressure*, ocasionando a passagem do *token* do *place StartTask* para o *place BloodPressureSensor*. Assim, enquanto de forma ativa a *transition*

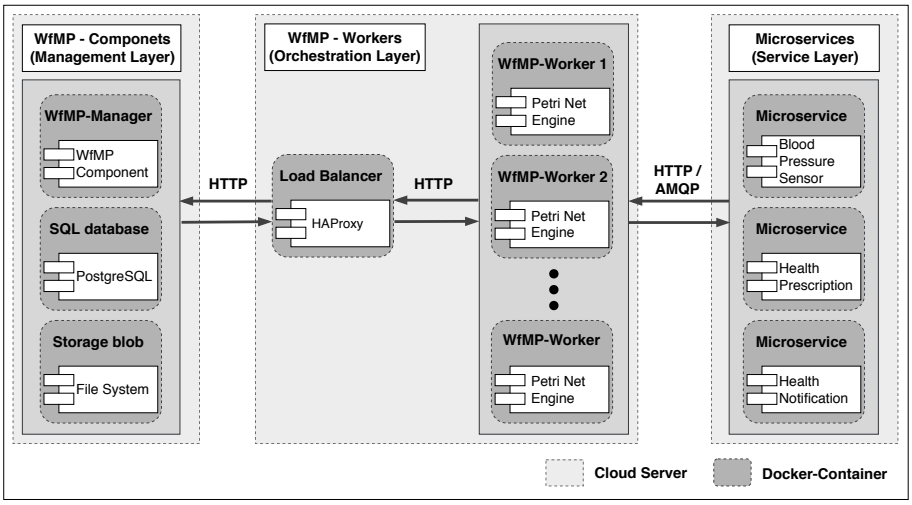

Figura 3.10: *Componentes que Realizam a Orquestração dos Workflows.*

getBloodPressure invoca o serviço, de forma passiva o valor da medida da pressão arterial é aguardado no place *BloodPressureSensor*. Esse mecanismo se repete durante todo o processo de orquestração.

3.5 Considerações Finais

Este capítulo apresentou as principais características da plataforma construída neste trabalho, com destaque para a arquitetura, a descrição e a implementação dos componentes e, considerando ainda, a implantação de serviços *web* de saúde na WfMP. Inicialmente, foi apresentado por meio da arquitetura um fluxo de atividades que representa as possíveis interações entre os diferentes tipos de usuários da WfMP. Em seguida, diante das descrições dos componentes, foram detalhadas as funcionalidades consideradas essenciais para o funcionamento da plataforma. Posteriormente, foram apresentadas as informações da implementação de cada um desses componentes. Por fim, foram analisadas, sob diferentes perspectivas, as formas em que a plataforma realiza a orquestração dos serviços de saúde.

CAPÍTULO 4

Validação e Avaliação da Plataforma

Este capítulo apresenta os resultados do desenvolvimento da plataforma obtidos com a modelagem de cenários de saúde utilizando a DSL e os recursos disponibilizados pela plataforma. Foi analisada, com base em dois cenários de testes, a forma em que situações práticas que podem ser gerenciadas por um plano de cuidados são representadas na plataforma (Seção 4.1). Também foi verificado, com o uso de diferentes configurações de *workflows*, o consumo de recursos computacionais da WfMP em um ambiente de computação em nuvem (Seção 4.2).

4.1 Modelagem de Workflows na WfMP

Como forma de analisar o comportamento da WfMP diante de possíveis *workflows* usados na área da saúde, foram utilizados dois cenários para realizar o acompanhamento de informações de um paciente. O primeiro deles consiste em um processo de diagnóstico de hipertensão (Subseção 4.1.1) e o segundo consiste no processo de acompanhamento de um paciente com Diabetes Tipo 2 (Subseção 4.1.2).

4.1.1 Cenário de Diagnóstico de Hipertensão

O cenário considerado é apresentado na Figura 4.1, sendo ele uma diretriz para a construção de um *workflow* que define um diagnóstico de hipertensão [15]. O principal objetivo desse cenário consiste em diferenciar casos de pacientes que possuem *white coat hypertension* (também conhecida como a síndrome do jaleco branco) daqueles que de fato possuem hipertensão. A causa dessa investigação se deve ao fato de que fatores externos (e.g., ambiente clínico) podem influenciar nos níveis de pressão arterial. Como alternativa, parte do diagnóstico da doença pode ser realizada em um ambiente domiciliar de monitoramento.

Tendo em vista que os valores de pressão arterial podem ser coletados por meio de sensores fisiológicos utilizados pelo paciente em sua residência, alguns serviços podem ser utilizados para receber esses valores, processá-los e tomar algum tipo de decisão.

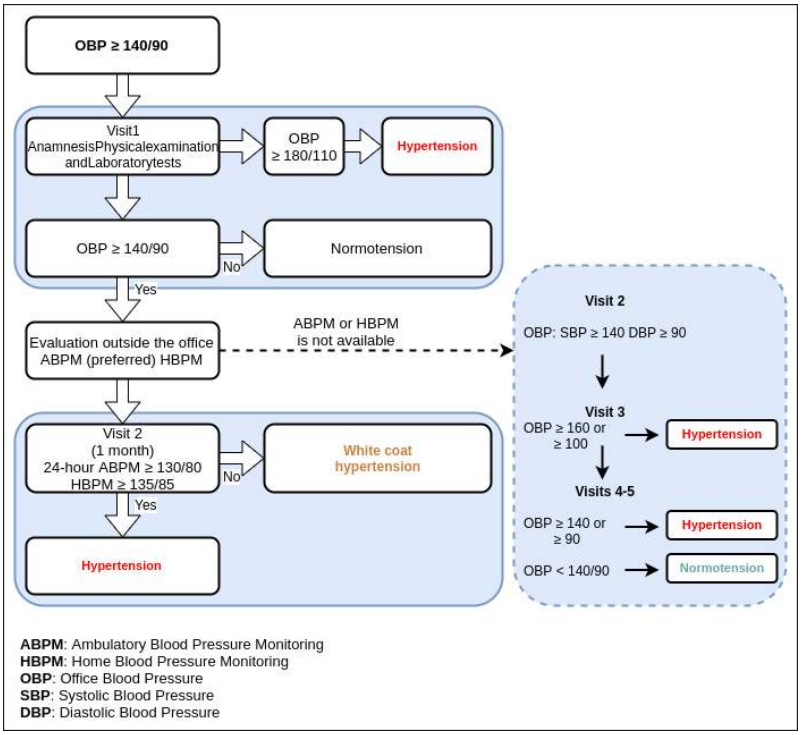

Figura 4.1: *Modelo de Diagnóstico para Hipertensão (Adaptado de [15]).*

Nesse sentido, foram considerados um serviço para captura dos dados coletados pelos sensores, um serviço para realizar o processamento e a persistência dos dados e um serviço de notificação para informar que o diagnóstico foi obtido ou algum problema foi detectado durante a avaliação. Considerando que esses serviços são desacoplados, mas podem interoperar utilizando padrões de representação de dados de saúde (e.g., OpenEHR) ou representações estruturadas registradas na WfMP, eles podem ser orquestrados utilizando a DSL apresentada neste trabalho.

Uma possível orquestração desses serviços é apresentada na Figura 4.2 por meio de um *workflow* e alguns trechos de código da DSL que foram utilizados para gerá-lo. Nesse *workflow* foram utilizados quatro *places*, o *BloodPressureSensor*, o *HealthcareAnalytics*, o *PatientHistory* e o *HealthProfessional*. O *BloodPressureSensor*, cuja representação usando a DSL é apresentada na parte superior da figura, possui um *case* com um evento que determina quando (por meio da primitiva *onReceive*) as ações são executadas (por meio da primitiva *call*). Nesse caso, as ações são executadas quando a plataforma recebe um novo *BloodPressure*, ou seja, um novo valor da pressão arterial de um paciente que esteja vinculado ao *workflow*.

Dessa forma, quando o valor é recebido, paralelamente são realizadas uma chamada ao serviço *analyze* do *HealthcareAnalytics* e uma outra chamada ao serviço *register* do *PatientHistory*. Portanto, em termos da Rede de Petri ocorre um *AND-split*, fazendo com que o *token* seja propagado para o *place HealthcareAnalytics* e para o

place PatientHistory. Como não existem transições oriundas do *place PatientHistory*, o processamento dessa ramificação da rede é finalizado. Já no *place HealthcareAnalytics*, cuja representação usando a DSL é apresentada na parte inferior da figura, é verificado, por meio da condição (*with*) *isTrue(HealthAssessment.diagnosis)*, se o diagnóstico do paciente foi obtido. Caso a condição seja verdadeira, é invocado um serviço de notificação ao profissional de saúde responsável, informando os resultados do processamento, sendo que isso é feito por meio da passagem do *token* ao *place HealthProfessional*. Caso a condição seja falsa, é executado um outro *case* cujo *script* foi omitido na figura, mas que redireciona o fluxo para o *place* inicial (*BloodPressureSensor*) a partir da chamada ao serviço *getBloodPressure*. Esse processo se repete até que o diagnóstico seja obtido.

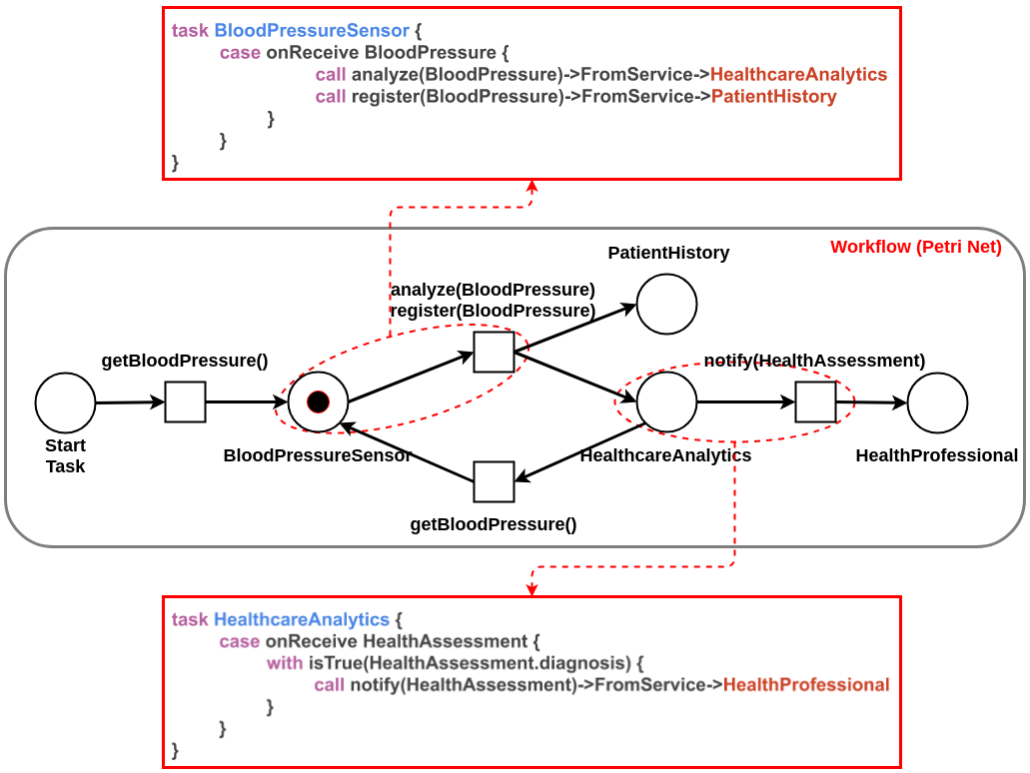

Figura 4.2: *Workflow construído a partir do Modelo de Diagnóstico de Hipertensão.*

Uma questão a ser considerada em relação ao *workflow* do exemplo apresentado é que ele poderia ser um *subworkflow* de um *workflow* mais amplo. Assim, caso após a obtenção de um diagnóstico seja verificado que o paciente de fato possui um problema de hipertensão, uma saída do *place HealthProfessional* poderia ser ligada a um outro *workflow* para o tratamento da doença, que por sua vez poderia ser modelado de acordo com as especificidades do problema. Portanto, uma generalização da solução apresentada neste trabalho refere-se ao uso DSL e, consequentemente da WfMP, para orquestrar quaisquer conjuntos de tarefas de um plano de cuidados, desde que essas tarefas possam ser implementadas em um formato de serviços e que, entre esses serviços, seja possível

estabelecer um nível de interoperabilidade para viabilizar a comunicação.

4.1.2 Cenário para Acompanhamento de Tratamentos de Diabetes Tipo 2

Este cenário de teste considera uma situação em que uma pessoa portadora de Diabetes Tipo 2 [50] é continuamente acompanhada com o auxílio de elementos computacionais que contribuem para um melhor gerenciamento dos procedimentos relacionados ao seu tratamento. A *American Diabetes Association* possui um conjunto de recomendações para o gerenciamento do tratamento de pacientes com Diabetes Tipo 2 que estão sintetizadas em um *pathway* representado por meio de um algoritmo [4].

Para representar o fluxo completo do gerenciamento das informações dos pacientes, esse algoritmo possui uma grande quantidade de passos, e esses passos, por sua vez, podem ser quebrados em um conjunto de outros passos. Para estes testes foram considerados os procedimentos relacionados ao gerenciamento do estilo de vida do paciente, ou seja, do passo *lifestyle management* do algoritmo apresentado em [4].

Embora no algoritmo mencionado anteriormente não haja um detalhamento em relação à forma em que o gerenciamento do estilo de vida do paciente deve ser realizado, em [48] é apresentado um modelo chamado *self-management education* (SME) para aumentar o engajamento de pacientes com Diabetes em relação ao processo de gerenciamento do tratamento da doença. Essa abordagem considera o uso de tecnologias baseadas em envios de SMS como ferramenta de apoio no gerenciamento de troca de mensagens entre pacientes e profissionais de saúde.

O modelo SME por si só possui um conjunto fluxos que, em um elevado nível de abstração, representa as principais diretrizes de um processo colaborativo centrado no paciente. Algumas etapas do modelo, como engajamento (*engage*), estimativas (*assess*), planejamento (*plan*), implementação (*implement*), monitoramento (*monitor*) e avaliação (*evaluate*) podem ser modeladas como *workflows*. Usando a DSL apresentada neste trabalho, essas etapas poderiam ser constituídas de vários *subworkflows* que implementam atividades que oferecem suporte direto ao paciente. Uma vez que o modelo SME incentiva o uso de tecnologias de comunicação por meio de notificações como forma de melhorar a interação entre paciente e profissional de saúde, o *workflow* correspondente, apresentado na Figura 4.3, define um mecanismo de notificações ao paciente e interessados em relação ao horário e aos procedimentos de prescrições de um medicamento.

O *workflow* da Figura 4.3 representa um encadeamento de tarefas que consiste em buscar uma prescrição de um medicamento às doze horas e encaminhá-la para um serviço de notificação (*notify*) do paciente (*Patient*) e para um serviço de envio de dados de prescrições (*sendMessage*) para usuários interessados (e.g., familiares). Esse *workflow*

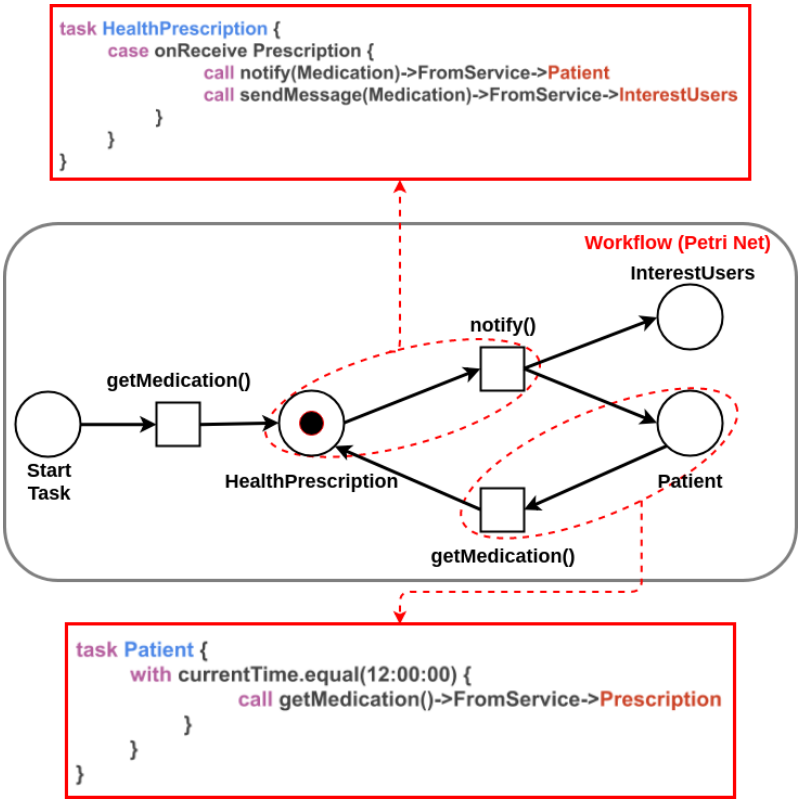

Figura 4.3: *Workflow para notificação de prescrições.*

poderia ser um dos *subworkflows* na etapa de implementação (*implement*) do modelo SME, pois consiste de um conjunto de tarefas que são realizadas em um estágio do gerenciamento de saúde do paciente em que ele já foi avaliadado e um plano de cuidados já foi definido.

4.2 Consumo de Recursos Computacionais da WfMP

Com o objetivo de analisar o comportamento da WfMP com respeito ao consumo de recursos necessários para viabilizar o conjunto de abstrações que ela oferece, nesta seção são descritos aspectos de configuração, execução e análise de experimentos em um ambiente de computação em nuvem. Na Subseção 4.2.1 estão as características do ambiente computacional utilizado, na Subseção 4.2.2 são descritos alguns experimentos realizados e na Subseção 4.2.3 estão os resultados obtidos.

4.2.1 Configuração do Ambiente

Foram utilizados quatro nós físicos, um nó para executar o *Manager* (*WfMP-Manager*), um nó para executar os *workers* (*WfMP-Worker*), um nó para executar micros-serviços externos da plataforma e um último para executar o *middleware* de gerência de filas de mensagens. Cada um dos nós utiliza separadamente uma CPU Intel Broadwell de

quatro núcleos. Enquanto o nó *WfMP-Worker* possui 26 GB de RAM os demais nós utilizam 8 GB de RAM. Cada nó também executa o sistema operacional Debian GNU/Linux 9 (stretch). Esses nós representam máquinas virtuais pertencentes ao Google Compute Engine[1] do provedor de serviços de nuvem Google Cloud Platform.

Uma questão importante a ser considerada na configuração do ambiente está relacionada à forma como ocorre o processamento paralelo de *workflows* na plataforma. Considerando que os *workflows* são expostos pela WfMP como serviços Web acessados por meio de APIs REST, cada solicitação de execução de um *workflow* corresponde a uma requisição ao servidor *web*. Além disso, a WfMP é executada em um servidor Apache Tomcat[2], que por sua vez implementa as especificações de um *Java Servlet*. Dessa forma, cada requisição HTTP feita para WfMP é processada independentemente usando um *pool* de *threads* gerenciado pelo *container* do *servlet*. Assim, definir o dimensionamento desse *pool* de *threads* é um fator importante para limitar a quantidade de requisições simultâneas suportadas pela plataforma.

O dimensionamento ideal de um *pool* de *threads* depende da natureza das tarefas que são executadas. Como regra geral, esse dimensionamento depende da relação entre o tempo de espera e o tempo de CPU e o número de CPUs disponíveis [26], sendo expresso pela seguinte relação:

$$Nthreads = Ncpu * (1 + W/C)$$

Nessa expressão (*Nthreads*) representa a quantidade de *threads*, *Ncpu* a quantidade de *CPUs* disponíveis e *W/C* a relação entre o tempo de espera (W) e o tempo estimado para o computador processar uma tarefa (C). Assim, na WfMP o tamanho dos *pools* de *threads* foram determinados experimentalmente a partir da análise do uso de recursos computacionais utilizados. Nessa análise foi considerado um tamanho inicial do *pool* proporcional à quantidade de núcleos da *CPU*.

Considerando que solicitar a execução de um *workflow* corresponde a realizar uma requisição ao servidor *web*, a relação a seguir foi usada para determinar a quantidade de threads para lidar com cada requisição separadamente:

$$Nthreads = NrequestServer * TrequestProcessing$$

Nessa expressão (*Nthreads*) representa a quantidade de *threads*, *NrequestServer* a quantidade servidores disponíveis e *TrequestProcessing* o tempo médio de processamento das requisições. É importante considerar que essa relação não escala linearmente devido a uma série de fatores de um sistema *multithreads* em Java, como, troca de contexto, interrupções do *garbage collector*, execução de outros processos na máquina física, etc. No entanto, ela foi utilizada como um direcionamento para a realização dos experi-

[1] https://cloud.google.com/compute/
[2] http://tomcat.apache.org/

mentos.

Vale também considerar que em situações em que a quantidade de requisições para execução de *workflows* é maior que o tamanho do *pool* de *threads*, alguns *workflows* podem ficar em uma fila de espera aguardando o processamento de outros *workflows*. No entanto, essa situação pode ser indesejada em cenários onde há processamento de dados críticos (e.g. notificação de uma situação emergencial de saúde). Dessa forma, uma possível solução viável é atribuir prioridades a cada uma das *threads* do *pool*. No entanto, a *ThreadPoolTaskExecutor API*[3] do *Spring Framework* utilizada na WfMP não oferece suporte para controlar as prioridades dos *pools* de *threads*. Assim, para fins de testes, foram atribuídos valores para o tamanho do *pool* que são sempre maiores ou iguais à quantidade de requisições de *workflows* solicitados paralelamente.

4.2.2 Configuração dos Experimentos

Dos quatro nós utilizados na WfMP, o *worker* foi determinante, por ser o responsável pelo processamento dos *workflows*. Diante disso, foi analisado o comportamento dos componentes da WfMP executados neste nó em relação ao consumo de recursos diante de cenários que demandam por um aumento de carga na plataforma. Foi verificado o comportamento dos *workers* com as seguintes configurações de *workflows*:

- *workflows* sequenciais
- *workflows* com ramificações
- *workflows* com *subworkflows*

As instâncias dos *workflows* sequenciais e *workflows* com ramificação usadas nos testes estão representadas na Figura 4.4, respectivamente, pelo Wf01 e Wf02. Já as instâncias dos *workflows* com *subworkflows* usadas para avaliar os *workers* estão representadas na Figura 4.5, respectivamente, pelo Wf03, SWf01 e SWf02. Para cada um desses casos foram realizados testes de carga e analisados o consumo de recursos dos componentes da plataforma.

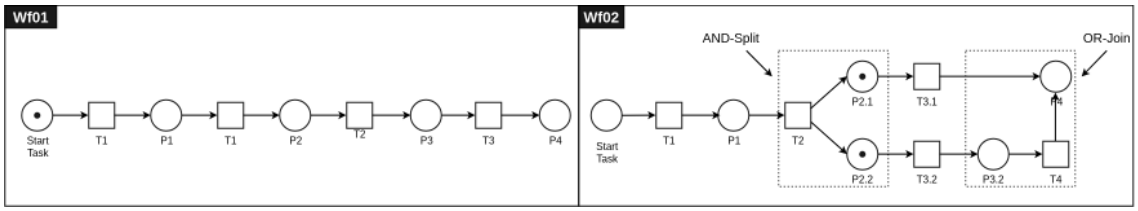

Figura 4.4: *Workflows usados nos testes (respectivamente, sequencial e com ramificação).*

[3] https://docs.spring.io/spring/docs/4.2.x/spring-framework-reference/html/scheduling.html

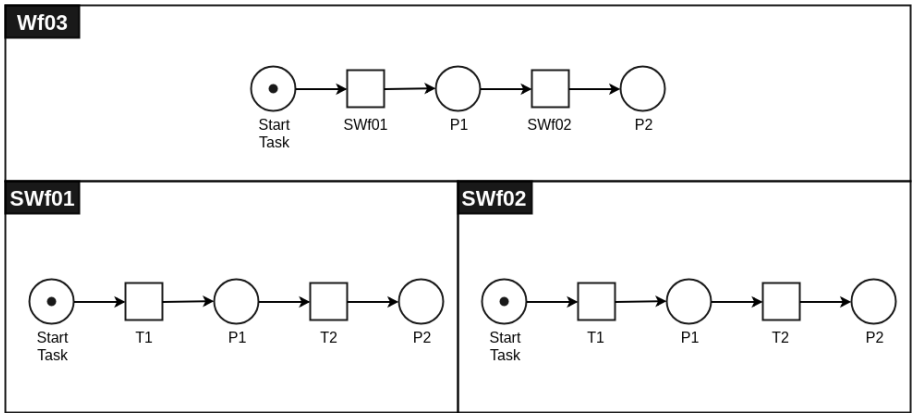

Figura 4.5: *Workflows usados nos testes (subworkflows).*

4.2.3 Análise do Consumo de Recursos

Workflow sequencial

A Figura 4.6 e a Figura 4.7 apresentam os resultados da variação do uso de recursos computacionais para o processsamento de instâncias do *workflow* sequencial Wf01, representado no gráfico da Figura 4.4. Para isso foi avaliado o consumo de *CPU* e memória RAM de acordo com o aumento do número de *workflows* sendo executados paralelamente na plataforma.

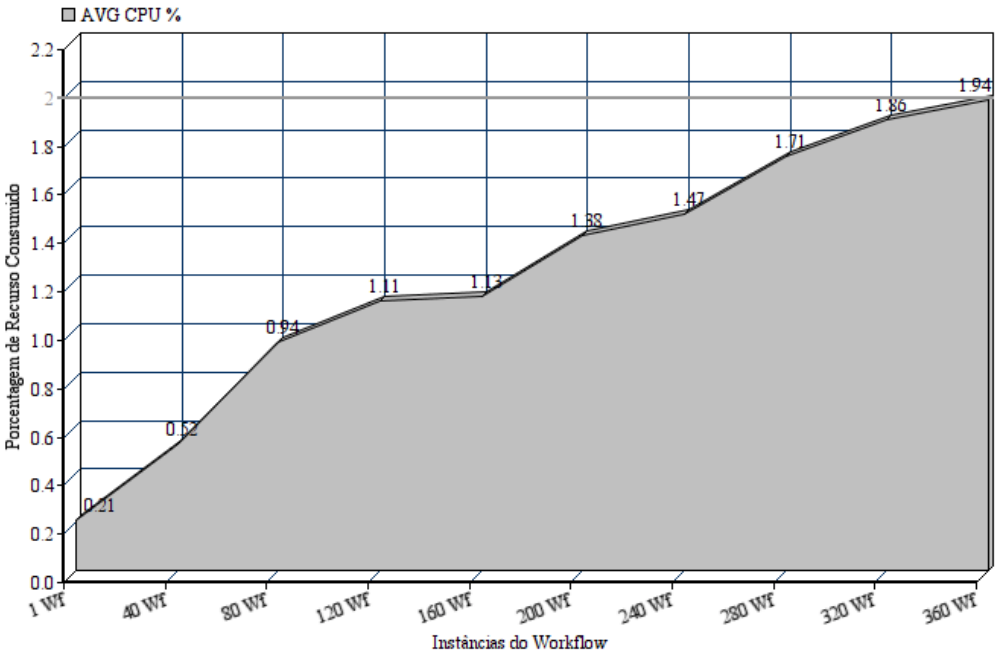

Figura 4.6: *Consumo de CPU pelas instâncias do workflow Wf01.*

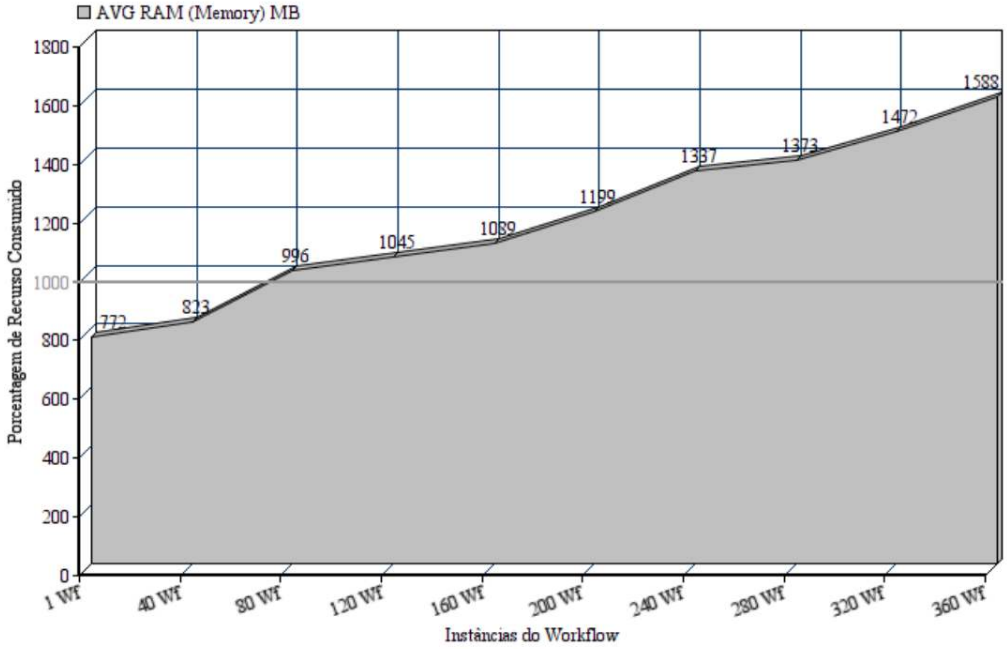

Figura 4.7: *Consumo de memória pelas instâncias do workflow Wf01.*

Workflow com ramificações

O *workflow* Wf02 (Figura 4.4) é uma variação do *workflow* Wf01, possuindo a adição de uma ramificação decorrente do *And-Split* na transição T2. Dessa forma, a partir dessa transição o *workflow* é processado em duas ramificações, sendo cada uma delas executada em uma *thread* diferente. Como pode ser observado na Figura 4.8 e na Figura 4.9, embora não houve uma sobrecarga em relação ao consumo de memória à medida em que são processadas várias instâncias desse *workflow* com ramificações, há um crescimento considerável do consumo de *CPU*. Uma questão a ser considerada é que o desvio-padrão em relação ao consumo de *CPU* é alto, de forma que em determinadas etapas do processamento dos *workflows* o consumo atinge 100%, enquanto em momentos de ociosidade esse valor é inferior a 1%. Para entender a causa desse aumento de consumo da *CPU*, a Figura 4.10 apresenta uma comparação do aumento da quantidade de processos ativos nos *workflows* Wf01 e Wf02.

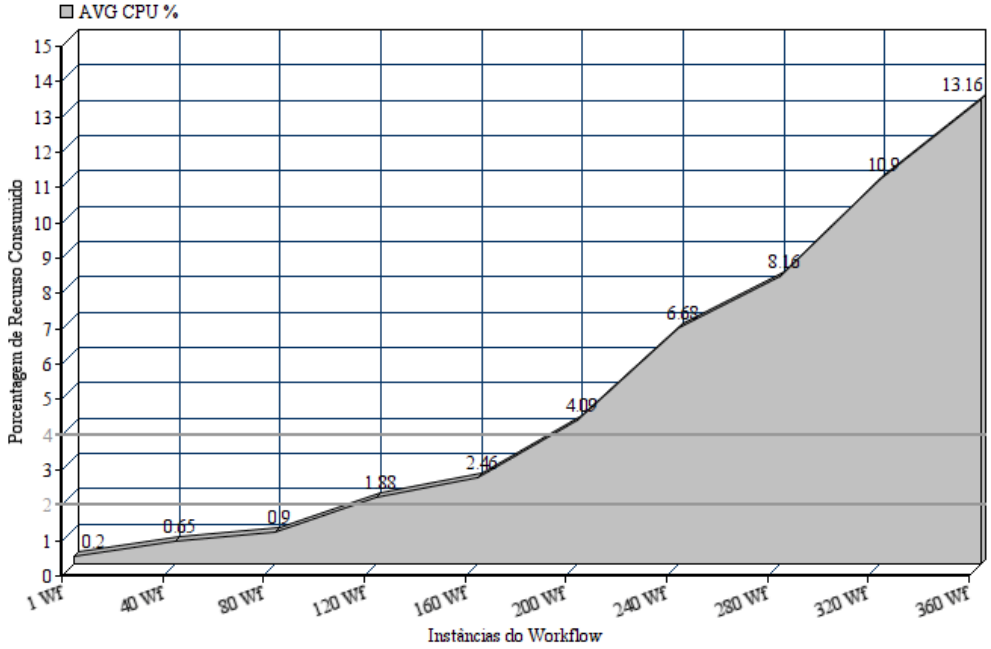

Figura 4.8: *Consumo de CPU pelas instâncias do workflow Wf02.*

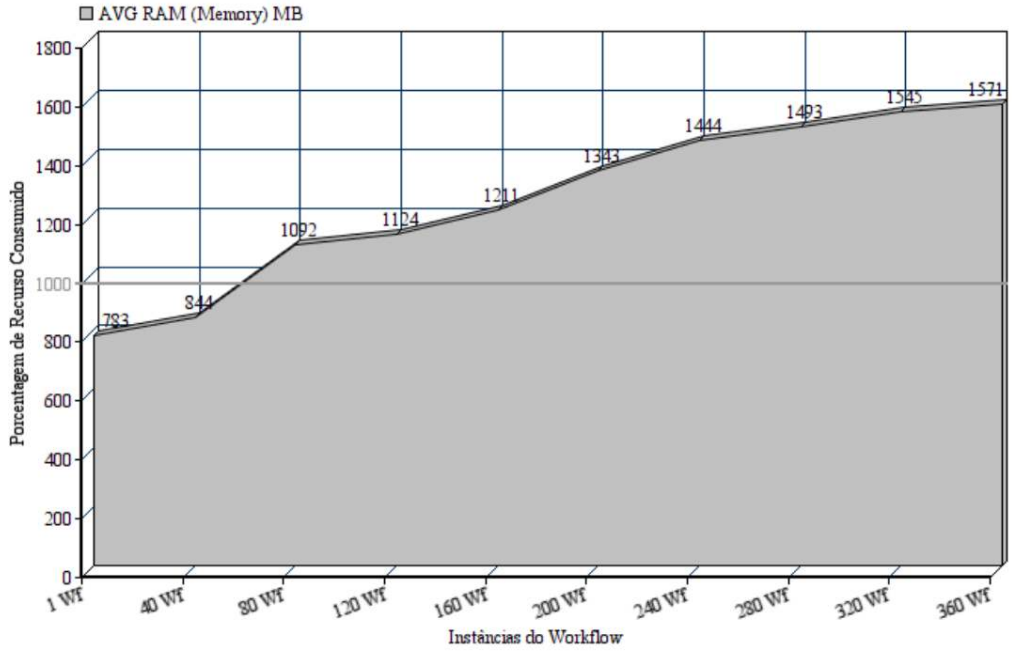

Figura 4.9: *Consumo de memória pelas instâncias do workflow Wf02.*

Como pode ser observado na Figura 4.10, a quantidade de processos ativos cresce em uma proporção maior no Wf02, se comparado com o Wf01. Logo, também haverá um crescimento na quantidade de instruções de controle a serem processadas pela *CPU*. Um dos fatores que impulsiona a elevada quantidade de processos ativos está relacionado ao processo de carregamento das representações dos *workflows* e das informações dos

serviços de um banco de dados relacional para um modelo de objetos (mapeamento objeto-relacional).

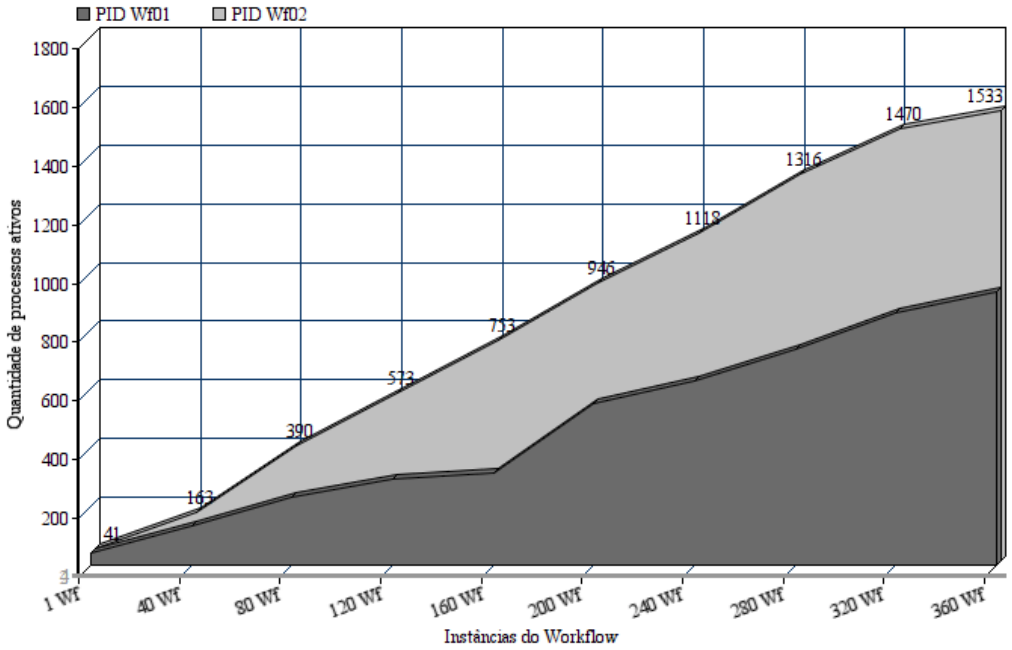

Figura 4.10: *Quantidade de processos ativos em relação ao número de instâncias de workflows.*

Workflow com *subworkflows*

A Figura 4.11 apresenta uma análise da variação do uso de recursos computacionais para processsamento do *workflow* Wf03 (Figura 4.5) com seus respectivos *subworkflows*, SWf01 e SWf02. Diante das questões mencionadas anteriormente, para esta análise foi avaliado apenas o consumo de memória RAM.

Em relação ao consumo de memória, ao comparar as três configurações de *workflows*, as duas primeiras (sequencial e com ramificação) obtiveram comportamentos semelhantes. Em ambos os casos houve uma variação entre 3% e 6% para um intervalo de 1 a 360 *workflows* sendo executados paralelamente. No caso do último teste (*workflows* com *subworkflows*) essa variação foi um pouco diferente, onde o consumo de memória esteve entre 3% e 8% para um intervalo de 1 a 360 *workflows* sendo executados concorrentemente. Esse aumento se deve ao fato de que diferentemente das chamadas aos serviços externos, realizar chamadas aos *subworkflows* implica no processamento de outros *workflows* pela plataforma. Assim, ao executar 360 instâncias do *workflow* Wf03, são necessárias 1080 instâncias dos objetos que realizam o processamento das Redes de Petri. Dessa forma, a vantagem de viabilizar a estensibilidade dos *workflows* (*workflows* em que algumas tarefas são *subworkflows*) está relacionada ao custo de processar instâncias desse *workflow* pela plataforma.

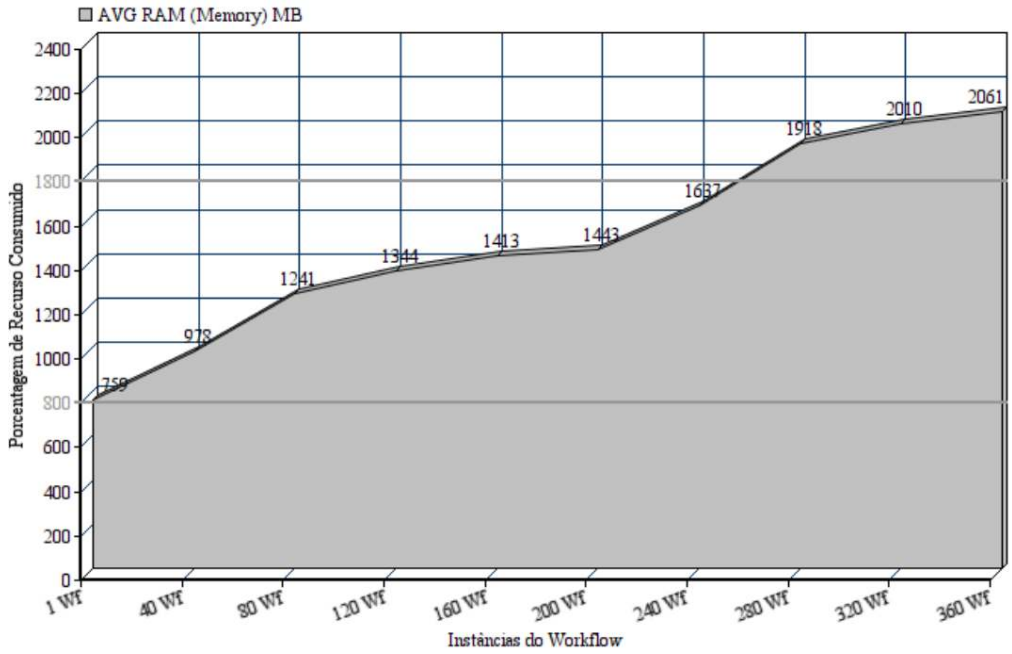

Figura 4.11: *Consumo de memória pelas instâncias do workflow Wf03.*

4.3 Considerações Finais

Este capítulo apresentou uma validação da plataforma por meio da modelagem de cenários de saúde utilizando a DSL e uma avaliação da plataforma em termos do uso de recursos computacionais em testes com diferentes configurações de *workflows*. Diante da modelagem dos cenários apresentados foi possível verificar durante a validação do trabalho a expressividade da DSL por meio da representação dos *workflows* e do uso dos microsserviços registrados na plataforma. Assim, foi realizado um mapeamento entre as características dos problemas de cada cenário para um conjunto de regras da DSL. Do ponto de vista de avaliação da plataforma, foi realizada uma análise de desempenho para verificar o comportamento de alguns componentes da plataforma diante da representação de diferentes configurações de *workflows* (e.g., sequenciais, com ramificações e com *subworkflows*). Como resultado dessa análise, foi verificado para cada um dos casos que não houve uma sobrecarga em termos de consumo de memória. No entanto, ao comparar *workflows* sequenciais e *workflows* com ramificações, foi observado uma sobrecarga em termos de consumo de *CPU*.

CAPÍTULO 5

Trabalhos Relacionados

Este capítulo apresenta os trabalhos relacionados na forma de uma discussão sobre a relação entre a plataforma apresentada neste trabalho e os processos de acompanhamento de pacientes (Seção 5.1), em relação a algumas linguagens atualmente utilizadas para composição de serviços (Seção 5.2), e por fim, sobre algumas ferramentas (*frameworks*/plataformas) utilizadas para composição de serviços (Seção 5.3).

5.1 *Homecare*, *Healthcare* e *Care Plan*

O conceito de Planos de Cuidados apresentado por Horta [31] e a proposta de continuidade em relação aos cuidados de saúde são amplamente trabalhados na área de ciências da saúde, buscando atingir diversos objetivos relacionados à melhoria da qualidade dos serviços em processos de saúde. Entre esses objetivos estão as melhorias nos fluxos de ações de educação em saúde, prevenção e postergação de doenças, cuidado precoce e reabilitação de agravos [53].

Um dos fatores que vêm impulsionando o uso da computação aplicada na área da saúde, e, mais especificamente, na linha de cuidados assistenciais de saúde está relacionado a propostas de continuidade no acompanhamento da saúde no ambiente domiciliar [28, 35]. Na literatura podem ser encontrados diversos trabalhos relacionados a esse assunto ao buscar por termos como *homecare*, *healthcare* e *care plan* [6, 14, 21, 49].

Do ponto de vista do uso de planos de cuidados (*care plan*) como parte do processo de acompanhamento da saúde de pacientes, há trabalhos que estão relacionados não apenas ao cenário em que o plano pode ser utilizado, mas também à forma em que essa ferramenta da enfermagem pode representar a descrição dos cuidados relacionados a um tratamento. Diante disso, trabalhos como [8] e [1] propõem soluções computacionais adaptativas/personalizáveis para representação do plano.

A proposta de Eric Browne apresentada em [8], possui diversas semelhanças com a plataforma apresentada neste trabalho. Alguns trabalhos de Browne exploram a problemática de modelar de planos de cuidados usando uma abordagem baseada em *workflows*. Ele ainda apresenta o conceito de *subworkflow* e discute amplamente

a possibilidade de lidar com requisitos de adaptabilidade/personalização de planos de cuidados. Nesse sentido, a principal diferença em relação a plataforma apresentada neste trabalho, se deve a implementação das características de orientação a serviços e da DSL para orquestração dos serviços na WfMP. Portanto, diante da abstração de serviço é possível lidar com o processamento de atividades individuais de um plano de cuidados e, por meio de mecanismos disponibilizados pela DSL, fazer o encadeamento dessas atividades.

Na literatura há diversos trabalhos que propõem representações formais para os planos de cuidados. Em [21] é apresentado um método formal que faz uso de autômatos cronometrados para representar possíveis cronogramas de atividades de plano de cuidados para pacientes monitorados em ambiente domiciliar. Esse trabalho propõe uma linguagem para representação de expressões temporais usadas no contexto de monitoramento de pacientes.

Outro trabalho que aborda técnicas de verificação formal de processos de planos de cuidados é apresentado em [6]. Nessa pesquisa é apresentada uma solução para lidar com processos dinâmicos e distribuídos no contexto de representação de tarefas relacionadas aos cuidados de saúde em ambientes domiciliares. Uma das características comuns desse trabalho com a WfMP é o uso do modelo de Redes de Petri para representação da semântica dos processos.

5.2 Linguagens para Composição de Serviços

O problema de construir *workflows* constituídos de serviços pode ser pensado como um problema de composição de serviços. Diante disso, existe um conjunto de linguagens, ferramentas e plataformas que pode auxiliar nesse processo. Analisando o estado da arte em relação ao desenvolvimento de sistemas de *workflows* orientados a serviços [51], as abordagens atualmente utilizadas são Web Service Business Process Execution Language (WS-BPEL ou BPEL), Web Service Choreography Interface (WSCI), Web Service Choreography Description Language (WS-CDL) e Ontology Web Language-Service (OWL-S).

Todas essas abordagens são direcionadas à composição de serviços *web* tradicionais, ou seja, serviços *web* que trabalham apenas sobre o protocolo SOAP (*Simple Object Access Protocol*). Isso é um problema, uma vez que atualmente grande parte dos serviços estão sendo desenvolvidos seguindo arquiteturas baseadas em REST (*Representational State Transfer*). No caso do BPEL, há uma proposta de extensão para que ele ofereça suporte a serviços *web* RESTful [42].

Analisando a linguagem BPEL como uma possível candidata para uso na modelagem de *workflows*, dadas as características do domínio deste trabalho, um dos primeiros

problemas identificados está relacionado ao nível de abstração em que as instruções são especificadas. BPEL pode ser vista como uma linguagem de baixo nível de abstração, se analisada do ponto de vista de uma ferramenta para auxiliar na modelagem de *workflows*, pois nela é utilizado um formato serializado em XML onde o usuário precisa se preocupar com uma série de questões em relação à comunicação dos serviços, tais como, a localização explícita dos serviços (ou do arquivo de descrição das interfaces dos serviços, WSDL) e questões relacionadas aos tipos primitivos dos dados utilizados nas interfaces dos serviços.

Considerando ainda que os recursos computacionais utilizados em um *workflow* podem fazer parte de uma arquitetura de microsserviços, outras questões precisam ser avaliadas. A principal delas é que quando se trata de uma arquitetura de microsserviços, a comunicação entre os microsserviços geralmente ocorre por meio de requisições assíncronas. Há uma série de fatores que favorecem a escolha de comunicação assíncrona nessas arquiteturas, sendo o principal deles a necessidade de lidar com requisitos de escalabilidade, disponibilidade e baixo acoplamento. Dessa forma, a troca de mensagens entre microsserviços geralmente ocorre por meio de filas de mensagens utilizando protocolos como AMQP. Esse tipo de comunicação é algo que nenhuma das abordagens citadas oferece suporte.

A linguagem para programação de microsservicos Jolie [27], embora tenha como principal objetivo lidar com questões relacionadas à distribuição e composição de microsserviços, não é uma linguagem específica de domínio. Por apresentar várias propriedades de uma linguagem imperativa, o nível de abstração é baixo em relação à especificação das instruções da linguagem, principalmente considerando as características dos usuários do domínio ao qual este trabalho se aplica.

5.3 Ferramentas para Composição de Serviços

Com o objetivo de identificar trabalhos que utilizam diferentes abordagens para coordenação e composição de microsserviços, foi realizada uma revisão bibliográfica da literatura. A partir dessa revisão foram encontrados trabalhos que apresentam algumas características comuns às que são propostas pela WfMP. Entre esses trabalhos estão, a Plataforma Medley [57], a ferramenta Microflows [40] e os *frameworks* Linkedator [47] e Skitter [45]. A principal semelhança encontrada em relação à WfMP é que todas utilizam alguma abordagem que busca facilitar a realização de composição de microsserviços. No entanto, existem algumas diferenças entre elas e a WfMP, principalmente se for levada em consideração a questão de especificidade de domínio. Uma análise comparativa desses trabalhos com a WfMP é apresentada na Tabela 5.1.

Tabela 5.1: *Comparação de outras Ferramentas com a WfMP*

	Orquestração de Serviços	**Abordagem**	**Domínio**
WfMP	Sim	DSL	Saúde
Medley	Sim	DSL	Composição de Serviços
Microflows	Sim	–	Composição de Serviços
Linkedator	–	OWL	Composição de Serviços
Skitter	Sim	DSL	*Reactive Workflows*

A plataforma Medley é a que apresenta uma maior quantidade de caraterísticas comuns à WfMP. Isso porque ambas as plataformas realizam coordenações de serviços a partir de orquestrações (ambas trabalham com o conceito de *workflow*) e oferecem uma DSL como uma forma de abstração para as tarefas relacionadas ao processo de composição. A principal diferença entre os dois trabalhos se deve ao domínio no qual eles estão inseridos. Enquanto o foco da plataforma Medley está diretamente relacionado à própria tarefa de composição de serviços, a WfMP tem como objetivo atender às necessidades de serviços de um domínio específico, ou seja, na WfMP a DSL e os seus procedimentos para viabilizar a comunicação entre os serviços têm como foco o domínio de saúde.

5.4 Considerações Finais

Este capítulo apresentou alguns trabalhos relacionados sob diferentes aspectos abordados nesta pesquisa. Foram analisados os trabalhos que possuem alguma intersecção com assuntos direcionados ao cenário de saúde, monitoramento remoto de pacientes e as diversas formas de representação de planos de cuidados. Em seguida, foram discutidas algumas relações entre as linguagens atualmente utilizadas para composição de serviços com a DSL desenvolvida neste trabalho. Por fim, foi feita uma análise comparativa entre algumas ferramentas (*frameworks*/plataformas) utilizadas para composição de serviços e a WfMP.

Lidar com especificidades do domínio de saúde é uma das principais características que difere a solução apresentada neste trabalho das demais encontradas na literatura. Seja pelas abstrações criadas pelos serviços ou pelas diretivas contidas na sintaxe da DSL, ao separar os fluxos de interação entre os diferentes tipos usuários da plataforma, a WfMP cria níveis de indireção que faz com que, no contexto de gerenciamento de atividades/tarefas dos planos de cuidados, ela se sobressaia em relação as outras ferramentas.

CAPÍTULO 6

Conclusões

Este capítulo apresenta as considerações finais, as principais contribuições e as limitações desta pesquisa, bem como trabalhos futuros.

6.1 Considerações Finais

Com o objetivo de viabilizar a construção de serviços de saúde especializados, visando possibilitar o desenvolvimento de sistemas computacionais de saúde personalizados para cada paciente, a *Workflow Management Platform (WfMP)* oferece um conjunto de ferramentas para modelagem, execução e gerenciamento de *workflows*. A plataforma permite que especialistas do domínio de saúde, diante de um conjunto de recursos computacionais, tenham condições de escolher aqueles adequados para o auxílio no tratamento de pacientes.

Nesta dissertação é apresentada uma arquitetura em que usuários com diferentes perfis (como profissionais de saúde e profissionais de computação) na plataforma têm diferentes perspectivas em relação à forma em que ocorrem as orquestrações. Enquanto profissionais de computação têm acesso, por exemplo, a APIs de manipulação de informações relacionadas a protocolos de comunicação e configurações de serviços, os profissionais de saúde têm acesso às informações dos serviços registrados, a formas de utilizá-los e a uma DSL para compor tarefas de um plano de cuidados com esses serviços.

A abstração proporcionada pela WfMP em relação ao direcionamento das responsabilidades dos usuários é um dos pontos importantes deste trabalho. Também foram apresentados alguns benefícios da abordagem baseada em *workflows* como serviços, tais como, extensibilidade e reutilização de serviços em tempo de execução. Esses são os benefícios que possibilitam a concepção de composição de serviços especializados de saúde a partir de serviços mais básicos.

A WfMP é uma plataforma candidata para realização de testes de integração de microsserviços de saúde utilizados para compor funcionalidades de planos de cuidados em ambientes de computação ubíqua. Dessa forma, poderão ser analisados cenários com

componentes que automatizam tarefas de um plano de cuidados adaptável/reconfigurável na forma de *workflows* concebidos e executados na plataforma.

6.2 Trabalhos Futuros

A seguir estão algumas possibilidades de trabalhos futuros com o foco no desenvolvimento de novas funcionalidades para a WfMP:

- desenvolvimento de ferramentas gráficas para auxiliar na construção de *Workflows*;
- análise do processamento de *workflows* considerando modelos computacionais aceitos na literatura que são baseados na propagação dos *tokens* de uma Rede de Petri a partir do uso de grafos de alcançabilidade;
- realização de um estudo de viabilidade para tornar a plataforma auto-escalável em tempo de execução;
- desenvolvimento de um componente para viabilizar a interoperabilidade entre os serviços a partir do uso de arquétipos do padrão OpenEHR.

Em relação à validação de tipos de dados de saúde, o processo atual utilizado na WfMP considera que os dados são representados usando o padrão *JSON Schema*[1], que consiste em representações em JSON para testar a validade de objetos também em JSON. No entanto, essa validação poderia ser realizada utilizando as representações de tipos de dados do padrão OpenEHR. Assim, como um trabalho futuro, essa modificação consistiria na substituição do componente de valiação de tipos de dados chamado Json-Schema na Figura 6.1 pelo componente OpenEHR Validator da Figura 6.2.

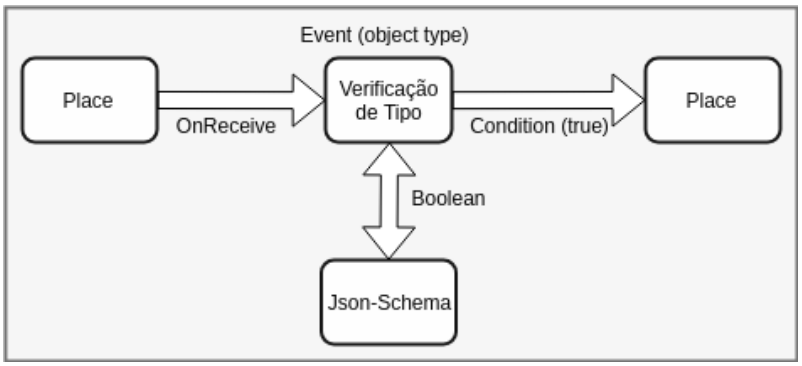

Figura 6.1: *Processo Atual de Validação de Tipos de Dados na WfMP.*

A Figura 6.3 apresenta um fluxo simplificado para realizar a validação de dados de saúde utilizando as representações disponibilizadas a partir de arquétipos do OpenEHR.

[1] https://json-schema.org/

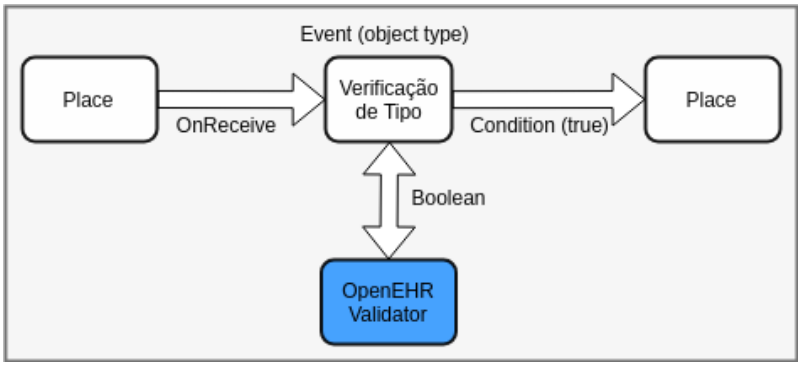

Figura 6.2: *Processo de Validação de Tipos de Dados na WfMP usando OpenEHR.*

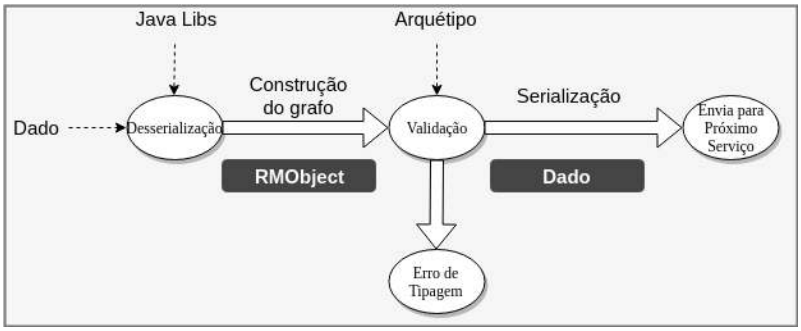

Figura 6.3: *Validação de Tipos de Dados no OpenEHR.*

Para encontrar a representação de dados primitivos de um tipo de dado representado usando o padrão OpenEHR, considere, por exemplo, um arquétipo para representação de dados do tipo *blood pressure*. Esse arquétipo possui um conjunto de atributos. O atributo systolic, por exemplo, é uma especificação de um ELEMENT chamado de `DV_QUANTITY` (informação contida na ADL do arquétipo). O `DV_QUANTITY` por sua vez possui os atributos representados nesse modelo de referência, como a magnitude (*double* que representa o valor da pressão sistólica), a unidade (ex. a *string* "mmhg") e a precisão. Para cada um dos outros atributos, existem tipos estruturados (que no OpenEHR são considerados tipos básicos). Esses tipos estão nos modelos de referência (MR) do padrão[2]. Há uma série de bibliotecas (no repositório oficial do OpenEHR) e ferramentas para manipular e fazer validação a partir desses esquemas.

[2]https://specifications.openehr.org/releases/RM/latest/index

Referências Bibliográficas

[1] ABIDI, S. S.; CHEN, H. **Adaptable personalized care planning via a semantic web framework**. In: *20th International Congress of the European Federation for Medical Informatics (MIE 2006), Maastricht*. Citeseer, 2006.

[2] AMARAL, N. D.; CUNHA, M. C. B.; LABRONICI, R.; OLIVEIRA, A. S. B.; GABBAI, A. A. **Assistência domiciliar à saúde (home health care): sua história e sua relevância para o sistema de saúde atual**. *Rev Neurociencias*, 9(3):111–17, 2001.

[3] ATZORI, L.; IERA, A.; MORABITO, G. **The internet of things: A survey**. *Computer networks*, 54(15):2787–2805, 2010.

[4] BA-ESSA, E. M.; ABDULRHMAN, S.; KARKAR, M.; ALSEHATI, B.; ALAHMAD, S.; ALJOBRAN, A.; ALDIJWI, A.; ALHAWAJ, A. **Closing gaps in diabetes care: From evidence to practice**. *Saudi journal of medicine & medical sciences*, 6(2):68, 2018.

[5] BAE, J.; BAE, H.; KANG, S.-H.; KIM, Y. **Automatic control of workflow processes using eca rules**. *IEEE transactions on knowledge and data engineering*, 16(8):1010–1023, 2004.

[6] BARKAOUI, K.; HICHEUR, A.; KHELDOUN, A.; LIU, D. **Modelling and analyzing home care plans using high-level petri nets**. In: *2016 13th International Workshop on Discrete Event Systems (WODES)*, p. 284–290. IEEE, 2016.

[7] BOGNER, J.; ZIMMERMANN, A. **Towards integrating microservices with adaptable enterprise architecture**. In: *Enterprise Distributed Object Computing Workshop. 20th International*, p. 1–6. IEEE, 2016.

[8] BROWNE, E. D.; SCHREFL, M.; WARREN, J. R. **Goal-focused self-modifying workflow in the healthcare domain**. In: *Proceedings of the 37th Annual Hawaii Int. Conference on*, p. 10–pp. IEEE, 2004.

[9] CAFFREY, C.; SENGUPTA, M.; MOSS, A.; HARRIS-KOJETIN, L.; VALVERDE, R. **Home health care and discharged hospice care patients: United states, 2000 and 2007**. *Natl Health Stat Report*, 38:1–27, 2011.

[10] CAMILLI, M.; BELLETTINI, C.; CAPRA, L.; MONGA, M. **A formal framework for specifying and verifying microservices based process flows**. In: *International Conference on Software Engineering and Formal Methods*, p. 187–202. Springer, 2017.

[11] CARPENITO-MOYET, L. J.; THORELL, A.; GARCEZ, R. M.; AMBROSINI, L. **Planos de cuidados de enfermagem e documentação: diagnósticos de enfermagem e problemas colaborativos**. Artmed, Porto Alegre (RS), 4 edition, 2006.

[12] CARVALHO, S. T.; COPETTI, A.; LOQUES FILHO, O. G. **Sistema de computação ubíqua na assistência domiciliar à saúde**. *Journal Of Health Informatics*, 3:2, 2011.

[13] CARVALHO, S. T.; LOQUES, O.; MURTA, L. **Dynamic variability management in product lines: An approach based on architectural contracts**. In: *Software Components, Architectures and Reuse (SBCARS)*, p. 61–69. IEEE, 2010.

[14] CHI, H.; CHOW, W.; CHUI, K.; MAN, K.; HANCKE, G. P. **A remote moniotring patient homecare gateway supporting streaming vital sign monitoring**. In: *IECON 2013-39th Annual Conference of the IEEE Industrial Electronics Society*, p. 8415–8419. IEEE, 2013.

[15] CLOUTIER, L.; DASKALOPOULOU, S. S.; PADWAL, R. S.; LAMARRE-CLICHE, M.; BOLLI, P.; MCLEAN, D.; MILOT, A.; TOBE, S. W.; TREMBLAY, G.; MCKAY, D. W.; OTHERS. **A new algorithm for the diagnosis of hypertension in canada**. *Canadian Journal of Cardiology*, 31(5):620–630, 2015.

[16] DEEN, M. J. **Information and communications technologies for elderly ubiquitous healthcare in a smart home**. *Personal and Ubiquitous Computing*, 19(3-4):573–599, 2015.

[17] DENG, M.; PETKOVIC, M.; NALIN, M.; BARONI, I. **A home healthcare system in the cloud–addressing security and privacy challenges**. In: *2011 IEEE 4th International Conference on Cloud Computing*, p. 549–556. IEEE, 2011.

[18] DOENGES, M. E.; MOORHOUSE, M. F.; GEISSLER, A. C.; DA CRUZ, I. C. F. **Planos de cuidado de enfermagem: orientações para o cuidado individualizado do paciente**. Guanabara-Koogan, Rio de Janeiro (RJ), 5 edition, 2008.

[19] ELHELW, M.; PANSIOT, J.; MCILWRAITH, D.; ALI, R.; LO, B.; ATALLAH, L. **An integrated multi-sensing framework for pervasive healthcare monitoring**. In: *Pervasive Computing Technologies for Healthcare, 2009. 3rd Int. Conference on*, p. 1–7. IEEE, 2009.

[20] FOWLER, M. **Domain-specific languages**. Pearson Education, 2010.

[21] GANI, K.; BOUET, M.; SCHNEIDER, M.; TOUMANI, F. **Using timed automata framework for modeling home care plans**. In: *2015 International Conference on Service Science (ICSS)*, p. 1–8. IEEE, 2015.

[22] GERMANO, E.; BATTISTI, D.; RIBEIRO, H. A.; CARVALHO, S. T. **Plano de cuidados ubíquo para acompanhamento domiciliar de pacientes**. In: *Congresso Brasileiro de Informática em Saúde (CBIS)*, p. 849–858, 2016.

[23] GERMANO, E.; SILVESTRE, B.; RIBEIRO, H. A.; CARVALHO, S. T. **Workflow management platform for orchestration of ubiquitous care plan services**. In: *Proceedings of the Euro American Conference on Telematics and Information Systems*, p. 42. ACM, 2018.

[24] GERMANO, E.; SILVESTRE, B. A.; CARVALHO, S. T. **Orquestração de serviços de um plano de cuidados ubíquo**. In: *Anais do XI Simpósio Brasileiro de Computação Ubíqua e Pervasiva (SBCUP)*. SBC, 2019.

[25] GHOSH, D. **Dsl for the uninitiated**. *Communications of the ACM*, 54(7):44–50, 2011.

[26] GOETZ, B.; PEIERLS, T.; LEA, D.; BLOCH, J.; BOWBEER, J.; HOLMES, D. **Java concurrency in practice**. Pearson Education, 2006.

[27] GUIDI, C.; LANESE, I.; MAZZARA, M.; MONTESI, F. **Microservices: a language-based approach**. In: *Present and Ulterior Software Engineering*, p. 217–225. Springer, 2017.

[28] HÄGGLUND, M.; CHEN, R.; KOCH, S. **Modeling shared care plans using contsys and open ehr to support shared homecare of the elderly**. *Journal of the American Medical Informatics Association*, 18(1):66–69, 2010.

[29] HAMADI, R.; BENATALLAH, B. **A petri net-based model for web service composition**. In: *Proceedings of the 14th Australasian database conference-Volume 17*, p. 191–200. Australian Computer Society, Inc., 2003.

[30] HOARE, C. A. R. **Monitors: An operating system structuring concept**. In: *The origin of concurrent programming*, p. 272–294. Springer, 1974.

[31] HORTA, W. D. A. **Processo de enfermagem**. In: *Processo de enfermagem*. EPU, 1979.

[32] HU, S.; HUANG, M.; FENG, W.; ZHANG, Y. **A smart health service model for elders based on eca-s rules**. In: *Software Engineering Research, Management and*

Applications (SERA), 2017 IEEE 15th International Conference on, p. 93–97. IEEE, 2017.

[33] IROJU, O.; SORIYAN, A.; GAMBO, I.; OLALEKE, J. **Interoperability in healthcare: benefits, challenges and resolutions**. *International Journal of Innovation and Applied Studies*, 3(1):262–270, 2013.

[34] KANG, K. C.; COHEN, S. G.; HESS, J. A.; NOVAK, W. E.; PETERSON, A. S. **Feature-oriented domain analysis (foda) feasibility study**. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.

[35] KOCH, S.; HÄGGLUND, M.; SCANDURRA, I.; MOSTRÖM, D. **Old@ home. technical support for mobile close care. final report**. *Education*, 4:14–8, 2005.

[36] LEE, W.; KAISER, G. E.; CLAYTON, P. D.; SHERMAN, E. H. **Ozcare: a workflow automation system for care plans.** In: *Proceedings of the AMIA Annual Fall Symposium*, p. 577. American Medical Informatics Association, 1996.

[37] LUPU, E.; DULAY, N.; SLOMAN, M.; SVENTEK, J.; HEEPS, S.; STROWES, S.; TWIDLE, K.; KEOH, S.-L.; SCHAEFFER-FILHO, A. **Amuse: autonomic management of ubiquitous e-health systems**. *Concurrency and Computation: Practice and Experience*, 20(3):277–295, 2008.

[38] MARTÍNEZ-COSTA, C.; MENÁRGUEZ-TORTOSA, M.; FERNÁNDEZ-BREIS, J. T. **An approach for the semantic interoperability of iso en 13606 and openehr archetypes**. *Journal of biomedical informatics*, 43(5):736–746, 2010.

[39] MURATA, T. **Petri nets: Properties, analysis and applications**. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[40] OBERHAUSER, R. **Microflows: Lightweight automated planning and enactment of workflows comprising semantically-annotated microservices**. In: *Proceedings of the Sixth International Symposium on Business Modeling and Software Design*, p. 134–143, 2016.

[41] PALESANDRO, A.; GUEGAN, C. G.; LACOSTE, M.; BENNANI, N. **Overcoming barriers for ubiquitous user-centric healthcare services**. *IEEE Cloud Computing*, 3(6):64–74, 2016.

[42] PAUTASSO, C. **Bpel for rest**. In: *International Conference on Business Process Management*, p. 278–293. Springer, 2008.

[43] RIBEIRO, H. A.; GERMANO, E.; CARVALHO, S. T.; ALBUQUERQUE, E. S. **Integrating social networks and remote patient monitoring systems to disseminate notifications**. In: *MEDINFO 2017: Precision Healthcare Through Informatics: Proceedings of the 16th World Congress on Medical and Health Informatics*, volume 245, p. 198. IOS Press, 2018.

[44] SABATÉ, E. **Adherence to long-term therapies: evidence for action**. World Health Organization, 2003.

[45] SAEY, M.; DE KOSTER, J.; DE MEUTER, W. **Skitter: a dsl for distributed reactive workflows**. In: *Proceedings of the 5th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems*, p. 41–50. ACM, 2018.

[46] SALIMIFARD, K.; WRIGHT, M. **Petri net-based modelling of workflow systems: An overview**. *European journal of operational research*, 134(3):664–676, 2001.

[47] SALVADORI, I.; HUF, A.; MELLO, R. D. S.; SIQUEIRA, F. **Publishing linked data through semantic microservices composition**. In: *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*, p. 443–452. ACM, 2016.

[48] SHERIFALI, D.; BERARD, L. D.; GUCCIARDI, E.; MACDONALD, B.; MACNEILL, G. **Self-management education and support**. *Canadian journal of diabetes*, 42:S36–S41, 2018.

[49] SOLANAS, A.; PATSAKIS, C.; CONTI, M.; VLACHOS, I. S.; RAMOS, V.; FALCONE, F.; POSTOLACHE, O.; PÉREZ-MARTÍNEZ, P. A.; DI PIETRO, R.; PERREA, D. N.; OTHERS. **Smart health: a context-aware health paradigm within smart cities**. *IEEE Communications Magazine*, 52(8):74–81, 2014.

[50] SURAMPUDI, P. N.; JOHN-KALARICKAL, J.; FONSECA, V. A. **Emerging concepts in the pathophysiology of type 2 diabetes mellitus**. *Mount Sinai Journal of Medicine: A Journal of Translational and Personalized Medicine: A Journal of Translational and Personalized Medicine*, 76(3):216–226, 2009.

[51] TAN, W.; ZHOU, M. **Service oriented workflow systems**. Hoboken, NJ, USA: Wiley/IEEE Press, 2015.

[52] VAN DER AALST, W.; VAN HEE, K. M.; VAN HEE, K. **Workflow management: models, methods, and systems**. MIT press, 2004.

[53] VERAS, R. **Linha de cuidado para o idoso: detalhando o modelo**. *Revista Brasileira de Geriatria e Gerontologia*, 19(6):887–905, 2016.

[54] VIEIRA, M. A.; CARVALHO, S. T. **(meta)modelagem de espaços inteligentes pessoais e espaços inteligentes fixos para aplicações ubíquas**. In: *Simpósio Brasileiro de Computação Ubíqua e Pervasiva (SBCUP)*, p. 1056–1065, 2016.

[55] WEISER, M. **The computer for the 21st century-scientific american special issue on communications**. *Computers, and Networks*, p. 94–104, 1991.

[56] WU, B.; DUBE, K. **Plan: a framework and specification language with an event-condition-action (eca) mechanism for clinical test request protocols**. In: *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*, p. 10–pp. IEEE, 2001.

[57] YAHIA, E. B. H.; RÉVEILLÈRE, L.; BROMBERG, Y.-D.; CHEVALIER, R.; CADOT, A. **Medley: An event-driven lightweight platform for service composition**. In: *International Conference on Web Engineering*, p. 3–20. Springer, 2016.

APÊNDICE A

Gramática da DSL

Representação da gramática da DSL usando a notação *Extended BackusNaur-Format* (EBNF) utilizada pelo Antlr4 no processo de construção do analisador sintático.

```
1
2  root : 'workflow' IDENTIFIER '{' firstPlaceDef otherPlaceDef+ '}';
3
4  firstPlaceDef : 'start task' '{' actionDef+ '}';
5
6  otherPlaceDef : 'task' IDENTIFIER '{' transitionDef+ '}';
7
8  transitionDef : 'case' eventDef '{' conditionDef '{' actionDef '}' '}'
9              | 'case' eventDef '{' actionDef '}'
10             ;
11
12 eventDef : eventType
13             | // epsilon
14             ;
15
16 conditionDef : 'with' expression
17             | // epsilon
18             ;
19
20 actionDef   : (targetDef callDef)+;
21
22 eventType   : 'onReceive' kind            // Asynchronous call
23             | 'onReturn'  kind endPoint   // Synchronous call
24             ;
25
26 kind        : primitive
27             | struct;
28
29 primitive   : 'integer' | 'decimal' | 'boolean' | 'string';
30
31 struct      : IDENTIFIER
32             | IDENTIFIER '.' struct;
33
34 expression  : expression AND expression      // Logical expression with and
35             | expression OR expression       // Logical expression with or
36             | statement                      // Logical functions
37             | comparison_expr                // Comparison expression
38             | LPAREN expression RPAREN       // Logical expression in parentheses
39             | logical_entity                 // Logical entity
```

```
40            ;
41
42  operand     : arithmetic_expr;
43
44  operator    : '<' | '>' | '<=' | '>=' | '=' | '<>';
45
46  arithmetic_expr : arithmetic_expr MULT arithmetic_expr  // Arithmetic expression
        multiplication
47            | arithmetic_expr DIV arithmetic_expr   // Arithmetic expression division
48            | arithmetic_expr PLUS arithmetic_expr  // Arithmetic expression addition
49            | arithmetic_expr MINUS arithmetic_expr // Arithmetic expression subtraction
50            | MINUS arithmetic_expr                 // Arithmetic expression negation
51            | LPAREN arithmetic_expr RPAREN         // Arithmetic expression with
                  parentheses
52            | numeric_entity                        // Numeric entity (variable)
53            ;
54
55  comparison_expr : operand operator operand   // Comparison expression with operator
56            | LPAREN comparison_expr RPAREN  // Comparison expression with parentheses
57            ;
58
59  logical_entity : (TRUE | FALSE)  // Logical constant
60          | IDENTIFIER             // Logical variable
61          ;
62
63  targetDef : 'target' IDENTIFIER
64          | // epsilon
65          ;
66
67  callDef  : syncCallDef 'call' endPoint
68        | syncCallDef 'call' endPoint '->' 'FromService' '->' struct // Change of place in
              Petri Net
69        | syncCallDef 'call' endPoint '->' 'FromWorkflow' '->' struct // Change of place
            in Petri Net
70        | syncCallDef 'end.workflow'
71        | syncCallDef 'end.workflow.export' struct
72        ;
73
74  syncCallDef : 'sync'
75        | // epsilon
76        ;
77
78  statement : booleanFunction LPAREN object RPAREN
79     | timeFunction '.' 'equal' LPAREN value RPAREN
80     | timeFunction '.' 'equal' LPAREN value (',' value)* RPAREN
81     | timeFunction '.' 'between' LPAREN value ',' value RPAREN
82     | 'waitTime' 'until' value // Waits for the specified time after reaching the state
83     ;
84
85  booleanFunction   : 'isTrue'
86        | 'isFalse';
87
88  timeFunction : 'currentTime'  // Compare the current time with the specified time
89        | 'currentDate'         // Compare the current date with the specified date
90        | 'currentDayWeek'      // Compare the current day of the week with the specified
              day
91        ;
92
```

```
value : NUMBER
   | object;

NUMBER : [0-9]+;

object : IDENTIFIER
   | IDENTIFIER '.' IDENTIFIER
   | endPoint;

endPoint : IDENTIFIER LPAREN RPAREN
   | IDENTIFIER LPAREN attribute (',' attribute)* RPAREN;

attribute : IDENTIFIER
   | IDENTIFIER '.' IDENTIFIER
   ;

DECIMAL : '-'?[0-9]+('.'[0-9]+)? ;

IDENTIFIER : [a-zA-Z_][a-zA-Z_0-9]* ;

TRUE     : 'true';
FALSE    : 'false';

LPAREN   : '(' ;
RPAREN   : ')' ;

MULT     : '*' ;
DIV      : '/' ;
PLUS     : '+' ;
MINUS    : '-' ;

AND      : 'and' ;
OR       : 'or' ;

numeric_entity : DECIMAL   // Numeric constant
         | IDENTIFIER      // Numeric variable
         ;

WS : [ \t\r\n]+ -> skip ;  // Skip spaces, tabs, newlines
```

APÊNDICE B

Procedimentos para Implantação da WfMP em um Servidor de Aplicações

B.1 Requisitos para Implantação do Sistema

Antes de realizar a implantação da plataforma, algumas ferramentas externas precisam ser instaladas e configuradas. Abaixo está uma lista com cada uma delas;

- Java Development Kit (JDK) na versão 8;
- Sistema de controle de versão Git[1];
- Ferramenta de automação de *build* Apache Maven[2];
- Sistema Gerenciador de Banco de Dados PostgreSQL na versão 9.4[3];
- Plataforma Docker[4];
- Ferramenta para definir a execução de múltiplos *containers* Docker-compose[5].

B.2 Processo de Implantação

Após configurar o ambiente necessário para iniciar os procedimentos de implantação da WfMP, alguns passos precisam ser seguidos para realização dessa tarefa. Esses passos consistem em obter o projeto que contém o código-fonte em um repositório com Git, configurar os arquivos com as informações necessárias para o carregamento da plataforma, compilar os projetos Maven, fazer o *Build* do projeto no Docker, restaurar a base de dados, configurar a ferramenta RabbitMQ e, por fim, fazer o executar os *containers* da WfMP na plataforma Docker. Nas subseções a seguir estão descritos cada um desses passos.

[1] https://git-scm.com/
[2] https://maven.apache.org/
[3] https://www.postgresql.org/
[4] https://www.docker.com/community-edition
[5] https://docs.docker.com/compose/

B.2.1 Obtenção do projeto

Os códigos-fontes da WfMP estão armazenados em um serviço de hospedagem de projetos chamado Bitbucket[6]. Assim, para obter o projeto da plataforma, no diretório de trabalho é preciso executar o comando:

$ git clone https://bitbucket.org/eliseu_germano/wfmp-cloud.git

Uma vez que se trata de um projeto *open-source* a ferramenta Git não solicita credenciais de acesso para fazer o *download* dos arquivos. Após finalizar o *download* é criado um diretório chamado wfmp-cloud contendo todos os artefatos da WfMP.

B.2.2 Configuração do projeto

Antes de compilar o projeto é necessário configurar algumas variáveis de ambiente. Essas variáveis estão nos arquivos application.properties de cada um dos projetos, sendo que no projeto *Manager* o caminho relativo é wfmp-cloud/WfMP-Manager/WfMS/src/main/resources/application.properties enquanto no projeto *Worker* é wfmp-cloud/WfMP-Worker/Worker/src/main/resources/application.properties. Para cada um dos projetos, as seguintes variáveis precisam ser definidas:

- **Database**
  - spring.datasource.url
  - spring.datasource.username
  - spring.datasource.password
- **RabbitMQ**
  - spring.rabbitmq.host
  - spring.rabbitmq.port
  - spring.rabbitmq.password
  - spring.rabbitmq.username
- **Server Host**
  - google.host.manager
  - google.host.worker
- **Local Environment**
  - server.port

Algumas configurações definidas nos arquivos application.properties precisam ser replicadas nos arquivos docker-compose.yml dos projetos *Manager* e *Worker*. Entre

[6] https://bitbucket.org/

essas configurações de ambiente estão as portas em que cada um dos projetos são executados no servidor e o nome do banco de dados criado para execução da plataforma.

B.2.3 Compilação do Projeto e Construção das Imagens

Após configurar o projeto, o mesmo pode ser compilado utilizando a ferramenta Maven. Isso deve ser feito de dentro do diretório raiz de cada projeto Maven, ou seja, nos diretórios wfmp-cloud/WfMP-Manager/WfMS e wfmp-cloud/WfMP-Worker/Worker. Dentro desses diretórios, utilizando os privilégios de superusuário, deve ser executado o seguinte procedimento para realizar a compilação:

- **# mvn clean install**

Com a finalização do processo de compilação sem nenhum erro, é possível realizar o *build* do projeto utilizado a plataforma Docker. Assim, dentro do diretório raiz de cada projeto (*Manager* e *Worker*), utilizando os privilégios de superusuário, deve ser executado os seguintes procedimentos:

- **# docker-compose build** (construção das *docker images*)
- **# docker-compose up -d** (construção dos *docker containers*)

B.2.4 Restauração da Base de Dados

Após a etapa compilação do projeto e construção das *docker images* é possível executar os *docker containers* da WfMP. No entanto, durante o processo de execução, alguns deles *container* devem apresentar problemas de acesso com a base de dados, pois a mesma precisa ser restaurada para que alguns componentes da plataforma funcione. Para fazer a restauração da base de dados deve ser realizado os seguintes procedimentos com privilégios de superusuário:

- **# docker-compose up -d** para executar o *container* que contém o PostgreSQL;
- **# docker ps** para listar os *containers* criados e copie o [CONTAINER_ID] do *container* que tiver o *name* wfmpworker_database_1;
- **# docker exec -it** [**CONTAINER_ID**] **bash** para se conectar ao *container* wfmpworker_database_1;
- Conectado ao *container* wfmpworker_database_1, acesse o SGBD PostgreSQL usando o comando **$ psql -U postgres**, crie um banco de dados com o comando SQL **CREATE DATABASE WfMP;**, crie um usuário chamado infadmin com o comando **CREATE USER infadmin SUPERUSER INHERIT CREATEDB CREATEROLE;** e coloque este usuário como um superusuário **ALTER USER infadmin WITH SUPERUSER;**;

- Execute o comando **# docker inspect** [**CONTAINER_ID**] **| grep -i ip** com o CONTAINER_ID do *container* que tiver o *name* wfmpworker_database_1 (e.g., **# docker inspect 823dd357462c | grep -i ip**). Este comando retornará em um dos campos do IPv6Gateway o campo IPAddress do *container* do banco de dados;
- Após obter o endereço IP do *container* a base de dados da WfMP pode ser restaurada. A restauração pode ser feito de diversas maneiras, como, por exemplo, por meio da ferramenta PgAdmin[7].

B.2.5 Configuração do RabbitMQ

Criar e executar instâncias do RabbitMQ é algo que pode der feito de diversas formas, como por meio pacotes específicos de cada sistema operacional, usando ferramentas em ambiente de computação em nuvem que trabalham com *RabbitMQ-as-a-Service* e pela imagem disponibilizada para plataforma Docker. Na WfMP foi utilizada uma *docker image* disponível no repositório oficial do RabbitMQ no Dockerhub[8]. Para isso foi criado em um arquivo docker-compose.yml onde foi definido um *service* com a *image* rabbitmq:3-management e duas portas para acesso ao RabbitMQ, sendo elas uma delas para acessar o Nó e outra para acessar a aplicação *Manager* do RabbitMQ. Seguindo essa abordagem, executar uma instância do RabbitMQ é algo que pode ser feito por meio de um único comando:

- **# docker-compose up -d**

B.2.6 Execução da WfMP na plataforma Docker

Com a base de dados restaurada e o RabbitMQ configurado, o próximo passo consiste em executar os *containers* da WfMP na plataforma Docker. Para isso, os seguintes procedimentos devem ser realizados no diretório raiz de cada projeto Maven (*Manager* e *Worker*):

- **# docker-compose down -d** (utilizado para parar algum *container* da WfMP, caso tenha algum em execução);
- **# docker-compose up -d** (etapa em que iniciará todos os *containers* de um projeto da plataforma);
- **# docker-compose logs** (para mostrar a saída retornada de cada um dos *containers*).

[7] https://www.pgadmin.org/
[8] https://hub.docker.com/