Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Sistema de gerenciamento de *workflows* M2P

Bruno de Oliveira Jucá

JUIZ DE FORA
DEZEMBRO, 2023

Sistema de gerenciamento de *workflows* M2P

Bruno de Oliveira Jucá

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Orientador: Eduardo Barrére

JUIZ DE FORA
DEZEMBRO, 2023

Sistema de gerenciamento de *workflows* M2P

Bruno de Oliveira Jucá

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Eduardo Barrére
Doutor em Engenharia de Sistemas e Computação

Eduardo Pagani Julio
Doutor em Computação

Carlos de Salles Soares Neto
Doutor em Informática

JUIZ DE FORA
05 DE DEZEMBRO, 2023

Resumo

Organizar o fluxo de dados dentro de uma aplicação pode ser uma tarefa custosa, capaz de consumir tempo e esforço na execução de projetos com o objetivo de realizar processamento de dados. Ferramentas como Workflow Management Systems são apresentadas como possíveis soluções para auxiliar na execução de *workflows* definidos com o objetivo de cumprir tarefas específicas. A partir de demandas encontradas no contexto do Laboratório de Aplicações e Inovação em Computação da UFJF, este trabalho busca encontrar aspectos relevantes em sistemas de gerenciamento de *workflows* que sejam capazes de contribuir para a construção de uma ferramenta personalizada, chamada de M2P, responsável por definir, interpretar e executar *workflows* em recursos computacionais disponíveis de maneira escalável, utilizando tecnologias atuais que simplifiquem processos intermediários na construção do sistema.

Palavras-chave: Workflow Management System, *workflow*, fluxo de dados, processamento de dados

Abstract

Managing dataflow within an application can be a high cost task, capable of consuming time and effort in the execution of projects aiming to performe data processing. Tools such as Workflow Management Systems are presented as possible solutions to aid in the execution of defined workflows with the aim of accomplishing specific tasks. From demands found in the context of the Laboratory of Applications and Innovation in Computing at UFJF, this work aims to find relevant aspects in workflows management systems that are capable of contributing to the construction of a personalized tool, called M2P, responsible for defining, interpreting and executing workflows in available computational resources in a scalable way, using new technologies that simplify intermediate processes in the development of the system.

Keywords: Workflow Management System, workflow, data flow, data processing

Agradecimentos

Agradeço a Deus, por me guiar durante todo meu caminho.

Agradeço aos meus pais, Adriana e Marco Aurélio†, por me darem apoio e por me incentivarem a seguir meus objetivos.

Agradeço ao meu irmão Marco Aurélio e à sua esposa Laís por todo apoio durante meu caminho acadêmico e pelo suporte durante a realização deste trabalho.

Agradeço aos meus familiares e amigos pelo apoio durante minha trajetória acadêmica.

Agradeço ao professor Eduardo Barrére pela orientação fundamental para a realização deste trabalho e pela competência em guiar grande parte de minha formação profissional.

Por fim, agradeço à UFJF e a todos os professores e funcionários que cruzaram meu caminho, possibilitando meu desenvolvimento pessoal e profissional.

"Se você já construiu castelos no ar, não tenha vergonha deles. Estão onde devem estar. Agora, dê-lhes alicerces."

Henry David Thoreau

Conteúdo

Lista de Figuras

Lista de Abreviações

DCC	Departamento de Ciência da Computação
UFJF	Universidade Federal de Juiz de Fora
WfMS	Workflow Management System
SWfMS	Scientific Workflow Management System
API	Application Programming Interface
REST	Representational State Transfer
MOM	Message-Oriented Middleware
WfMC	Workflow Management Coalition
DAG	Directed Acyclic Graph
WSFL	Web Services Flow Language
BPEL	Business Process Execution Language
ADL	Abstract Workflow Description Language
SPA	Single Page Application
RePesq	Rede Integrada de Pesquisa em Alta Velocidade
OCR	Optical Character Recognition
VAD	Voice Activity Detection
ASR	Automatic Speech Recognition

1 Introdução

1.1 Apresentação do tema

Grande parte do processamento de dados e de informações como um fluxo depende, atualmente, de ferramentas e de sistemas desenvolvidos para auxiliar tarefas computacionais complexas (CUGOLA; MARGARA, 2012). *Workflows* definem as sequências de tarefas necessárias para modelar um determinado processo, e um Workflow Management System (WfMS) é um sistema que tem como objetivo automatizar a interconexão dessas tarefas ao gerenciar suas execuções e suas trocas de informações (DEELMAN et al., 2018).

A utilização desses sistemas ajuda cientistas na realização de experimentos ao abstrair a gestão de recursos computacionais e ao permitir maior enfoque nos temas de pesquisa propriamente ditos. Além disso, a comunidade empresarial também se beneficia do uso desses sistemas em automação de processos internos, o que incentiva que iniciativas com interesses comerciais também abordem o tema (DEELMAN et al., 2009).

Workflow Management Systems geralmente oferecem interfaces gráficas que permitem que o usuário monitore e interfira em processos, além de possibilitar uma forma de interação mais simples, sem utilização de programação diretamente. Adicionalmente, *workflows* bem especificados, assim como a definição dos módulos que os compõem na realização de tarefas, permitem a replicabilidade de experimentos de uma forma confiável (KLUGE; FRIEDEL, 2018).

1.2 Contextualização

Nas últimas décadas, observou-se um aumento da necessidade de ferramentas direcionadas à análise de dados e ao aprendizado de máquina. Sistemas de *workflow* representam algumas dessas ferramentas (MITCHELL et al., 2019). No ambiente científico, os *workflows* se tornaram ferramentas essenciais e amplamente utilizadas para impulsionar pesquisas e descobertas (DEELMAN et al., 2018). Mitchell et al. (2019) apontam que WfMSs desen-

volveram maneiras de gerenciar a execução de tarefas em infraestruturas computacionais diversas, como em servidores locais, em *clusters*, em recursos computacionais de alta performance e até mesmo em plataformas populares de computação em nuvem. Tudo isso auxilia cientistas a executarem *workflows* complexos.

No contexto do Laboratório de Aplicações e Inovação em Computação (LApIC)[1], alguns trabalhos trouxeram a necessidade de apresentar uma arquitetura que controlasse a execução do fluxo de processamento, que é dividido em microsserviços (BARRÉRE; SOUZA; SOARES, 2020; JR. et al., 2020). Esses trabalhos utilizaram arquiteturas similares internamente, mas servindo aos propósitos individuais, sem modularidade para integração de novos microsserviços ou reutilização das partes integrantes de cada trabalho. Nesse sentido, observa-se a capacidade da abstração das arquiteturas similares, e essa abstração pode ser interpretada como um WfMS.

1.3 Descrição do problema

Organizar como deve se dar a transferência de dados e a organização estrutural de serviços especializados que realizam subtarefas pode ser custoso. Esse problema não é prioridade do ponto de vista do desenvolvedor que tem foco em produzir uma solução por meio de um algoritmo e não deseja se preocupar, necessariamente, com o desenvolvimento de uma arquitetura que realize a comunicação dos serviços (DEELMAN et al., 2009).

Ao examinar os trabalhos desenvolvidos no LApIC que apresentam arquiteturas similares, é possível perceber que houve necessidade de destinar tempo e esforço para elaborar a arquitetura de execução das tarefas. O fato de a mesma tecnologia ter sido utilizada, tendo sido capaz de entregar bons resultados e de se demonstrar escalável, permite encarar esse problema específico como uma necessidade de um WfMS customizado e simples que atenda a requisitos internos, algo observável em outros trabalhos (MITCHELL et al., 2019).

[1] https://www.ufjf.br/lapic

1.4 Justificativa

Apresentar um novo WfMS como uma solução que permita abstrair o funcionamento do encadeamento de tarefas entre módulos é importante para auxiliar a realização de processamentos de dados. Muitas das ferramentas existentes que buscam cumprir esse papel apresentam funcionalidades complexas e especificações pouco detalhadas de como usar cada uma delas. A proposta de um WfMS mais simples se apresenta como uma alternativa a essas ferramentas.

Este trabalho busca apresentar um WfMS que utilize novas tecnologias para facilitar o desenvolvimento da aplicação. Nesse sentido, é interessante apresentar uma alternativa às soluções existentes em termos de ferramentas que compõem o sistema e modos de utilização. Além de ser proposto como uma solução geral, o WfMS busca servir como ferramenta para uso do LApIC, uma vez que a demanda foi identificada a partir de trabalhos desenvolvidos pelo laboratório.

Um sistema capaz de organizar, detalhar e monitorar processamentos pode auxiliar desenvolvedores, pesquisadores e empresas que estejam interessados em utilizar esses recursos para processar informações em larga escala ou que estejam dispostos a contribuir com a comunidade por meio da disponibilização de módulos ou de *workflows* predefinidos que sirvam como ferramenta para outros usuários.

1.5 Objetivos

O objetivo geral do presente trabalho é apresentar um WfMS simples, com funcionalidades básicas de controle de execução e de monitoramento de um *workflow*. O trabalho busca apresentar o sistema como um *framework* que facilite a definição de *workflows* de propósitos gerais, de acordo com padrões específicos, e que possibilite fácil reprodutibilidade de experimentos, amplo reúso de módulos e escalabilidade.

De maneira específica, este trabalho busca:

- dar continuidade a pesquisas anteriores, explorando suas sugestões de trabalhos futuros e partindo de seus resultados;

- apresentar uma Application Programming Interface (API) que siga o modelo da arquitetura Representational State Transfer (REST) para utilização do sistema, assim como toda documentação necessária;
- disponibilizar uma interface gráfica para monitoramento do sistema que possibilite a execução de *workflows* sem necessidade de programação;
- definir guias que auxiliem no desenvolvimento de *workflows* e de módulos.

1.6 Metodologia

Com a finalidade de alcançar os objetivos estabelecidos neste trabalho, inicialmente foi realizada uma revisão da literatura. É importante apresentar definições já estipuladas que serviram como referências para a implementação de alguns dos recursos do sistema. Além disso, é importante identificar como trabalhos similares implementaram algumas das funcionalidades que o presente trabalho também apresenta. Diante de tecnologias emergentes e desses recursos, busca-se aplicar esses conceitos no desenvolvimento de um sistema que atenda aos requisitos do LApIC. Diante da apresentação do sistema, é necessário validar a implementação por meio de testes envolvendo trabalhos realizados no laboratório que se apliquem ao uso da ferramenta.

Diante disso, para a realização deste trabalho, foram propostas as seguintes etapas:

- revisão da literatura;
- análise dos trabalhos do LApIC e de seus requisitos em relação a um WfMS;
- seleção e estudo das tecnologias a serem utilizadas no desenvolvimento;
- implementação e documentação do *backend* do sistema;
- implementação do *frontend* do sistema;
- validação do sistema aplicado a um trabalho do LApIC;
- análise dos resultados.

A validação do sistema será realizada a partir do momento em que o trabalho selecionado for aplicado no ambiente da ferramenta, possibilitando a obtenção de seus resultados de processamento e posterior análise do impacto que a adaptação terá gerado na execução, considerando o que se espera de ferramentas dessa natureza.

2 Fundamentação teórica

Este capítulo apresenta alguns conceitos necessários para o entendimento do presente trabalho e de como se deu seu desenvolvimento baseado na literatura. A Seção 2.1 apresenta os conceitos de *workflow* e de WfMS. A Seção 2.2 define conceitos relativos à comunicação entre serviços, ao apresentar o conceito de Message-Oriented Middleware (MOM) e ao explicitar, especificamente, o Advanced Message Queuing Protocol (AMQP). A Seção 2.3 discorre sobre virtualização e aborda o uso de virtualização a nível de aplicação. Por fim, a Seção 2.4 apresenta as considerações finais sobre a fundamentação teórica apresentada.

2.1 Workflow Management Systems

De acordo com a Workflow Management Coalition (WfMC), um *workflow* preocupa-se com a automação de procedimentos em que informações ou tarefas são coordenadas entre participantes para atingir um objetivo de acordo com regras preestabelecidas (HOLLINGSWORTH; HAMPSHIRE, 1995). Os *workflows* são interpretados como um conector para sistemas distribuídos e embora a princípio fosse um conceito aplicado recorrentemente no ambiente de negócios, demonstrou-se interessante para a área de experimentos científicos. (BARKER; HEMERT, 2008).

Workflows também podem ser definidos como atividades que envolvem a execução coordenada de múltiplas tarefas, realizadas por diferentes entidades de processamento. Uma tarefa, por sua vez, seria determinado processamento que deve ser realizado sob alguma especificação concreta em certo programa. A especificação de um *workflow* envolve a descrição dos aspectos constituintes de cada tarefa que são relevantes para controle e execução, quais sejam as relações de dependência entre tarefas, suas entradas e suas saídas. A execução das múltiplas tarefas ou do *workflow* por diferentes entidades de processamento pode ser controlada por um *software* denominado WfMS (RUSINKIEWICZ; SHETH, 1995).

WfMSs são ferramentas para auxiliar os desafios de *workflows* experimentais. Os

WfMSs têm papel de coordenar a execução dos *workflows* do começo ao fim, agendando a execução das tarefas e fornecendo informações analíticas sobre as execuções. O objetivo desses sistemas é fornecer um ambiente especializado de programação que simplifique os esforços necessários para orquestrar processamentos e experimentos (DEELMAN et al., 2009).

Tradicionalmente, um *workflow* tem sido representado como um Directed Acyclic Graph (DAG) em WfMSs, em que os nós do grafo representam tarefas computacionais e as arestas representam dependências entre as tarefas. A abordagem comumente utilizada é usar algoritmos de escalonamento que disparem a execução de tarefas assim que suas dependências sejam satisfeitas (MITCHELL et al., 2019).

Em Mitchell et al. (2019), são apontados quatro casos de uso gerais de WfMSs:

- *workflows* científicos;
- *workflows* para análise de dados (incluindo Big Data e Machine Learning);
- *workflows* para sensores e Internet of Things (IoT);
- *workflows* relacionados ao setor comercial, de desenvolvimento e de negócios.

No meio científico, o conceito de *workflow* tem sido aplicado na automatização de experimentos em larga escala (e-Science). Cientistas necessitam de ferramentas que possibilitem a integração de componentes especializados na resolução problemas para atestar hipóteses científicas. *Workflows* científicos contribuem para um ambiente de novas descobertas por meio do gerenciamento, da análise, da simulação e da visualização de dados científicos (BARKER; HEMERT, 2008).

Análise de dados em massa e aplicações de aprendizado de máquina estão presentes em muitas pesquisas científicas recentes. *Workflows* destinados a essas áreas apresentam desafios particulares, em especial no que diz respeito ao modo com que os dados são gerenciados (MITCHELL et al., 2019).

Outro caso de uso orientado a dados que têm crescido é o de sistemas baseados em sensores. Nesse caso, os dados se apresentam amplamente distribuídos e há a necessidade de coordená-los de maneira eficiente. Além disso, é necessário processar informações com

base na chegada de novos dados vindos das fontes heterogêneas. Aplicações IoT, que têm ganhado popularidade, apresentam necessidades similares (MITCHELL et al., 2019).

Por último, no setor de desenvolvimento de *software* e no setor comercial, *workflows* têm sido importantes para automatizar tarefas complexas no ciclo de vida de um *software*. Empresas têm criado soluções próprias para lidar com seus problemas particulares, como é o caso do Apache Airflow[2], criado pelo AirBnb[3] (MITCHELL et al., 2019).

2.2 Message-Oriented Middleware

Os *softwares* e os sistemas de computação têm se apresentado cada vez mais de maneira distribuída. É interessante romper limites físicos e permitir que a comunicação entre diferentes máquinas possibilite a criação de sistemas mais complexos, usufruindo dos benefícios da escalabilidade horizontal. Dessa maneira, os sistemas podem apresentar diferentes componentes, com diferentes *hardwares*, sistemas operacionais e linguagens de programação. Nesses ambientes, é necessário assegurar estabilidade e disponibilidade dos serviços (CURRY, 2004).

O mecanismo de MOM fornece uma maneira de viabilizar a comunicação entre peças de um sistema de maneira facilitada. Muitos serviços distribuídos, em geral, dependem fundamentalmente de MOMs, que podem ser definidos como qualquer infraestrutura de *middleware* que disponibilize serviço de mensagens. Dessa maneira, os nós dentro de uma rede composta de entidades que compõem um sistema podem entregar e receber mensagens. Isso permite que sistemas flexíveis sejam criados, de maneira que as partes de um sistema sejam mais independentes umas das outras, desde que mantenham o padrão de comunicação (CURRY, 2004).

O AMQP é um protocolo assíncrono na camada de aplicação que disponibiliza comunicação por mensagem, enfileiramento e roteamento. O protocolo é apresentado como confiável e implementa recursos de segurança, afinal, foi inicialmente projetado para auxiliar em transações financeiras, que demandam esses recursos. O AMQP fornece uma

[2]https://airflow.apache.org
[3]https://www.airbnb.com

descrição detalhada do formato de dados na comunicação como um *stream* de *bytes*. A especificação detalhada do protocolo permite sua implementação em diversas linguagens (O'HARA, 2007).

Existem alguns conceitos fundamentais definidos no AMQP ilustrados na Figura 2.1. Um produtor pode ser um componente do sistema que publica mensagens em uma *exchange*. Uma *exchange*, por sua vez, envia as mensagens publicadas às filas, que estão conectadas a essa *exchange*. Finalmente, as filas enviam as mensagens para os consumidores. Essa arquitetura permite que haja uma independência entre produtores e consumidores. Como os produtores publicam em uma *exchange*, que assume a responsabilidade de encaminhar as mensagens, é relativamente fácil mudar algum comportamento sem que seja necessário alterar configurações nos serviços que enviam e recebem mensagens. A possibilidade de existirem múltiplos consumidores inscritos em uma fila permite alta escalabilidade (JOHN; LIU, 2017).

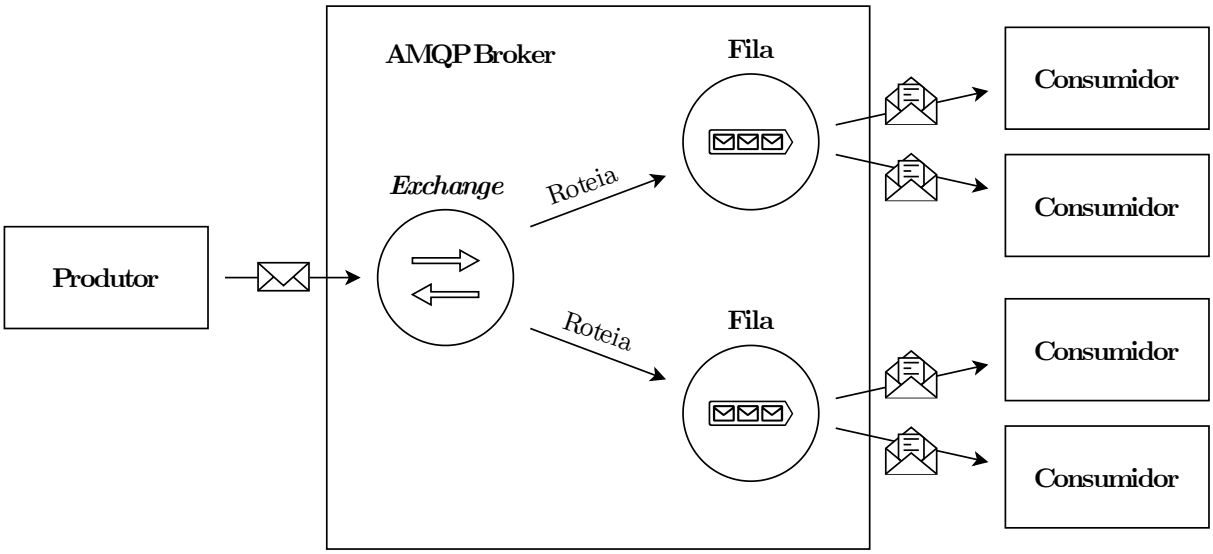

Figura 2.1: Funcionamento do protocolo AMQP como um MOM

O RabbitMQ[4] é um *message broker* de código aberto, um programa que administra a troca de mensagens definidas por protocolos formais entre outros programas. Ele permite que aplicações troquem mensagens por meio do protocolo AMQP. Dessa ma-

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

neira, funciona como uma implementação do protocolo e permite sua utilização prática (ROSTANSKI; GROCHLA; SEMAN, 2014).

2.3 Virtualização

Em ambientes distribuídos, principalmente naqueles baseados na nuvem, virtualização tem sido amplamente empregada. Como maneira de gerenciar os recursos computacionais disponíveis e destiná-los a usos apropriados, máquinas físicas podem fazer uso de *hypervisors* para administrar máquinas virtuais. *Hypervisors* são *softwares* capazes de emular sistemas operacionais visitantes em uma máquina hospedeira. Dessa maneira, é possível isolar partes integrantes de um sistema, possibilitando que diferentes aplicações sejam executadas em diferentes ambientes isoladamente. Entretanto, esse mecanismo executa múltiplos *kernels* em uma máquina física, o que tem um custo computacional alto (SCHEEPERS, 2014).

Uma alternativa a esse tipo de virtualização total seria a virtualização a nível de aplicação, também chamada de virtualização a nível de sistema operacional ou de virtualização com base em contêineres. Nesse tipo de virtualização, contêineres são estruturas que empacotam uma aplicação com todas as dependências necessárias para que ela seja executada. Os contêineres são executados com base em um único sistema operacional, sob gerenciamento de um *engine* que é responsável por conectá-los ao sistema operacional. Ao utilizar apenas um *kernel* e ao compartilhar os recursos da máquina física com todas as aplicações, essa alternativa de virtualização consome menos recurso da unidade de processamento central (CPU), menos memória e menos recursos de rede, fornecendo eficiência, escalabilidade e bom custo computacional. Sistemas distribuídos beneficiam-se diretamente da aplicação dessa técnica (WATADA et al., 2019).

A Figura 2.2 ilustra os aspectos relacionados a cada um dos tipos de virtualização abordados.

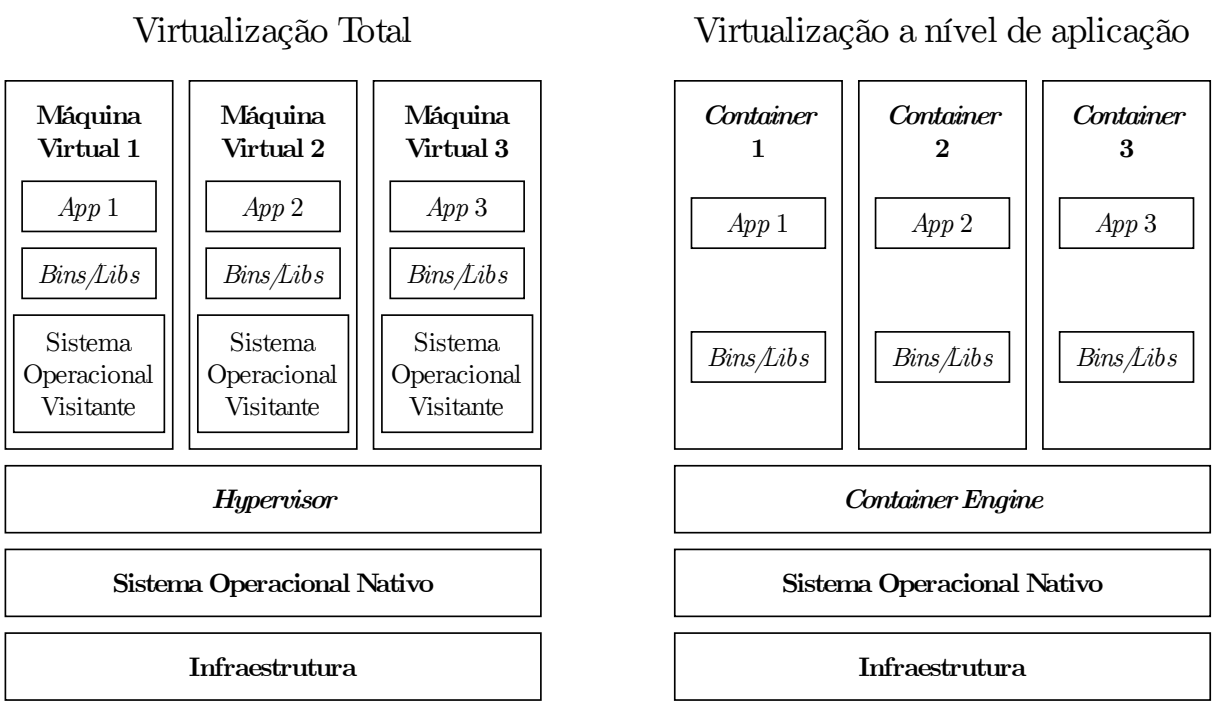

Figura 2.2: Comparação entre virtualização total e virtualização a nível de aplicação

Workflows científicos são, usualmente, executados em ambientes distribuídos devido à natureza dos problemas para os quais se busca solução. As tarefas que compõem um *workflow* precisam ser executadas em ambientes bem definidos para cada uma delas. Dessa forma, a migração de aplicações é um desafio, já que é necessário manter esses ambientes consistentes e também assegurar a execução independente do sistema operacional de migração e das aplicações instaladas nele. A utilização de contêineres soluciona esses problemas, tendo sido testada e validada como uma forma de isolar essas tarefas e facilitar a implantação dos *workflows* e de suas tarefas. Um contêiner é capaz de definir um ambiente isolado, seguro, replicável e escalável para essas aplicações, com todas as dependências necessárias para executá-las (ZHENG; THAIN, 2015).

O Docker[5] é uma plataforma que permite aplicar virtualização a nível de aplicação, por meio da utilização de contêineres. O Docker Engine é o *software* que controla os contêineres. O Docker especifica um arquivo de configurações, o Dockerfile e, por meio dele, é possível criar imagens Docker. A partir de uma imagem, é possível executar um ou mais contêineres, que empacotam uma aplicação. Desse modo, é possível proporcionar

[5]https://www.docker.com

uma maneira mais simples de lidar com problemas de migração e de escalabilidade de sistemas (MERKEL, 2014).

2.4 Considerações finais

Neste capítulo foram apresentados conceitos fundamentais que servem como embasamento teórico para o desenvolvimento do trabalho proposto. Foram apresentados conceitos básicos sobre WfMSs e sobre como eles são capazes de gerenciar a execução de *workflows*. Foi apresentado o conceito geral de MOM e o modo como o protocolo AMQP se encaixa como um *middleware* para troca de mensagens. Ainda foram apresentados conceitos de virtualização e, mais especificamente, de virtualização baseada em contêineres, mostrando como essa ferramenta pode ser útil para proporcionar isolamento, escalabilidade e reprodutibilidade em *softwares*.

3 Trabalhos relacionados

Este capítulo descreve trabalhos relacionados que apresentam soluções para o gerenciamento de *workflows*. Cada um dos trabalhos analisados propõe um WfMS ou um *framework* de funcionamento similar. São discutidas as características de cada um deles quanto à modelagem representativa de *workflows* e quanto à escolha de como gerenciar suas execuções. Ainda se busca descrever as soluções de interface gráfica propostas e as funcionalidades principais presentes em cada um dos sistemas. As seções seguintes estão divididas por trabalho, descrevendo cada um deles.

Os trabalhos apresentados nesta seção foram selecionados a partir de uma busca por artigos que apresentam uma ferramenta de gerenciamento de *workflow*, sendo que o termo *WfMS* serviu como termo de busca. A única exceção é o Apache Airflow, que embora não tenha sido proposto em um artigo, já foi citado como uma solução para o gerenciamento de *workflows* (MITCHELL et al., 2019). Foram selecionados trabalhos que apresentam relevância acadêmica e que ao mesmo tempo apresentam características que se relacionam com o objetivo desta pesquisa.

3.1 Apache Airflow

O Apache Airflow[6] é uma plataforma de código aberto para gerenciamento de *workflows*. De acordo com o *site* do projeto, ele foi iniciado pelo Airbnb e passou pela Apache Incubator antes de ser oficializado como um projeto principal da Apache em 2019. O programa é popular entre usuários interessados em realizar análise de dados.

O programa é escrito, principalmente, em Python, e os *scripts* que definem os *workflows* também são escritos nessa linguagem. Um *script* de *workflow* é denominado um DAG, em referência a um grafo acíclico direcionado, que está, essencialmente, presente em todos os sistemas gerenciadores de *workflows*.

O fato desses *scripts* de configuração serem definidos em Python permite que as

[6]https://airflow.apache.org

configurações sejam realizadas por meio de código, e os desenvolvedores possam, também, importar bibliotecas que ajudem a definir determinadas especificidades. Entretanto, isso pode limitar o uso da ferramenta quando analisado sob a perspectiva de que poderia ser útil para usuários que não tenham tanta habilidade em programação e que estejam mais interessados no resultado dos processamentos.

Os operadores constituem os blocos fundamentais dos *workflows* do Airflow, já que representam uma tarefa dentro de um *workflow*. O *software* fornece uma variedade de operadores padrões para tarefas comuns.

O gerenciador de dependência de tarefas da aplicação garante a execução das tarefas na ordem correta de acordo com suas dependências, possibilitando execuções sequenciais ou em paralelo. Além disso, o sistema fornece mecanismos de agendamento e de execução automática baseada em eventos externos (*triggering*).

O Airflow fornece uma interface *web* centralizada na qual os usuários podem monitorar o *status* das tarefas em execução e visualizar os respectivos *logs*. Também é fornecida uma interface de linha de comando para integração com aplicações externas.

O sistema garante escalabilidade, assegurando ser possível executar *workflows* de larga escala com milhares de tarefas. Suporta execução de tarefas distribuídas e possibilita a implantação em modo *cluster*, para alta disponibilidade e tolerância a falhas. Possibilita escalabilidade horizontal com a adição de *workers*, realizando o balanceamento de carga das tarefas.

A Figura 3.1 mostra uma das páginas da interface gráfica do Apache Airflow.

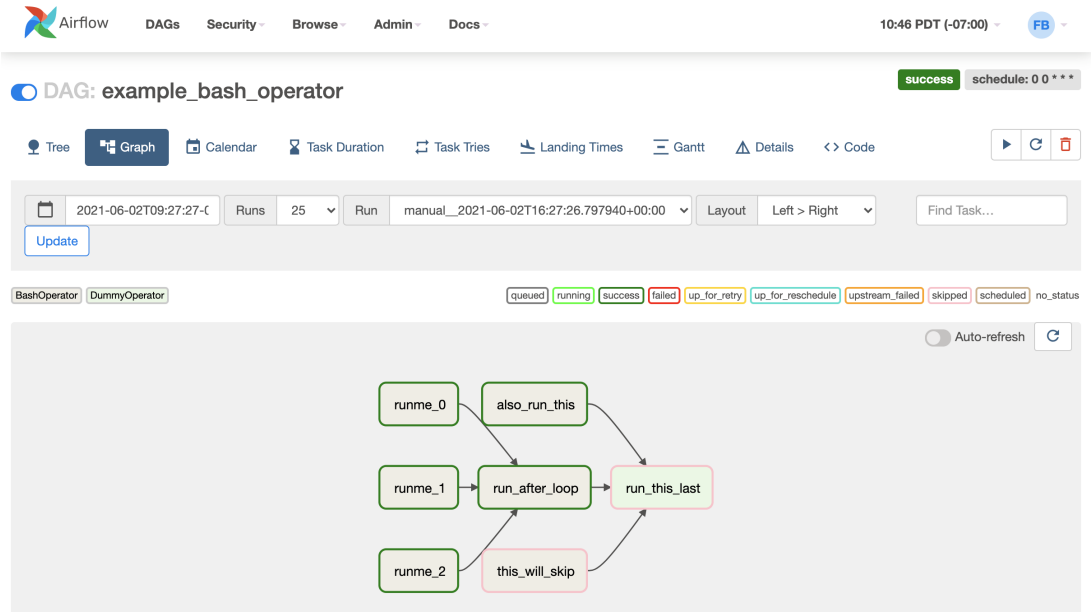

Figura 3.1: Página de DAGs no Apache Airflow. Imagem disponível no site do projeto.

3.2 Watchdog

O Watchdog é um WfMS feito para a automação de análise de dados experimentais em larga escala. Dentre os recursos e as funcionalidades presentes no *software*, destacam-se a facilidade no processamento de dados que se encontram replicados, o suporte para execução em ambientes distribuídos, a detecção de erros customizável e a possibilidade de intervenção manual em uma execução se necessário (KLUGE; FRIEDEL, 2018).

O sistema é implementado em Java, o que os autores argumentam tornar a plataforma independente de sistema operacional, facilitando o compartilhamento de recursos entre sistemas, ou seja, tanto de módulos como de *workflows*.

O Watchdog fornece uma interface gráfica para a construção de *workflows*, usando os módulos predefinidos do sistema, e também fornece *scripts* auxiliares para a criação e definição de novos módulos.

A execução de *workflows* é feita por meio da interface gráfica ou por meio de uma interface de linha de comando. Adicionalmente, há uma interface *web* para auxiliar no monitoramento do *status* de execução e para possibilitar que o usuário interfira no processamento em caso de erros.

O sistema apresenta dois *workflows* padrões para testes e que também servem de validação do sistema. Um deles é simples e utiliza ferramentas básicas de manipulação de arquivos, geralmente pré-instaladas em sistemas Unix, para compactar e extrair dados. O outro ilustra o potencial de execução de *workflows* mais complexos, por meio de um experimento envolvendo diferenciação de sequenciamento genético que utiliza alguns *softwares* externos.

A Figura 3.2 apresenta uma tela da interface gráfica do Watchdog.

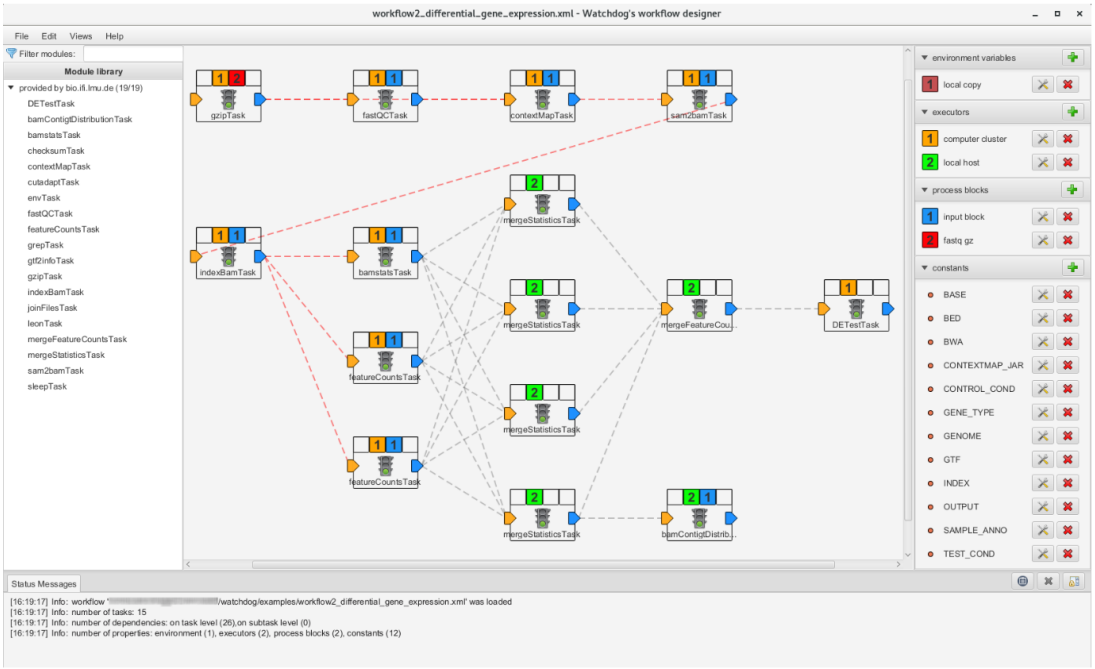

Figura 3.2: Página de construção de *workflows* no Watchdog (KLUGE; FRIEDEL, 2018).

3.3 Pegasus

Em Deelman et al. (2015) os autores propõem o Pegasus, um Sistema de Gerenciamento de Workflow Científico (Scientific Workflow Management System ou SWfMS) de código aberto, desenvolvido, principalmente, em Java, desde 2001. O sistema é capaz de orquestrar a execução de tarefas e de monitorar o andamento da execução. O Pegasus é integrado através de APIs atualmente oferece suporte para Java, Perl, Python e R.

Além de acesso pelas APIs, o sistema também se encontra encapsulado em al-

gumas plataformas que o utilizam para gerenciar execuções como CyVerse[7], HUBzero (MCLENNAN; KENNELL, 2010) e Wings (GIL et al., 2011). O CyVerse se propõe a disponibilizar recursos e ferramentas permitindo a colaboração de cientistas e de suas ferramentas apresentadas para manipulação e análise de dados.

O Pegasus utiliza uma linguagem de domínio específico chamada Abstract Workflow Description Language (ADL) para descrever os *workflows* em forma de DAGs. A linguagem permite definir as tarefas, suas dependências e seus dados de entrada e de saída. Ainda oferece suporte a outras linguagens, como a Workflow Description Language e a Common Workflow Language. A execução dos *workflows* definidos é otimizada com base nos recursos computacionais disponíveis. Desse modo, o *workflow* é analisado, e o *software* faz considerações sobre localização dos dados e recursos disponíveis para fazer a otimização da execução.

Quanto ao gerenciamento de dados, o Pegasus é capaz de automaticamente mover os dados diante dos recursos disponíveis usando protocolos confiáveis de transferência de arquivos, garantindo integridade dos dados. Além disso, faz integração com algumas bases de dados definidas em seu catálogo.

O Pegasus gerencia a execução dos *workflows* coordenando diferentes *engines* de execução, como Condor, HTCondor e Docker. Ele coleta estatísticas de execução que disponibiliza ao usuário. A aplicação ainda pode ser estendida com o uso de *plugins*.

O sistema é construído para suportar processamento distribuído em *workflows* de larga escala, escalando horizontalmente ao possibilitar execução paralela. Ainda fornece mecanismos de tolerância a falhas como reexecução de tarefas e *checkpoints*.

O artigo que apresenta o sistema mostra um estudo de caso em que a aplicação é utilizada para testar um *workflow* que executa processamento de informações sísmicas em larga escala.

Na Figura 3.3 é possível visualizar uma das páginas da interface gráfica do Pegasus.

[7]https://www.cyverse.org

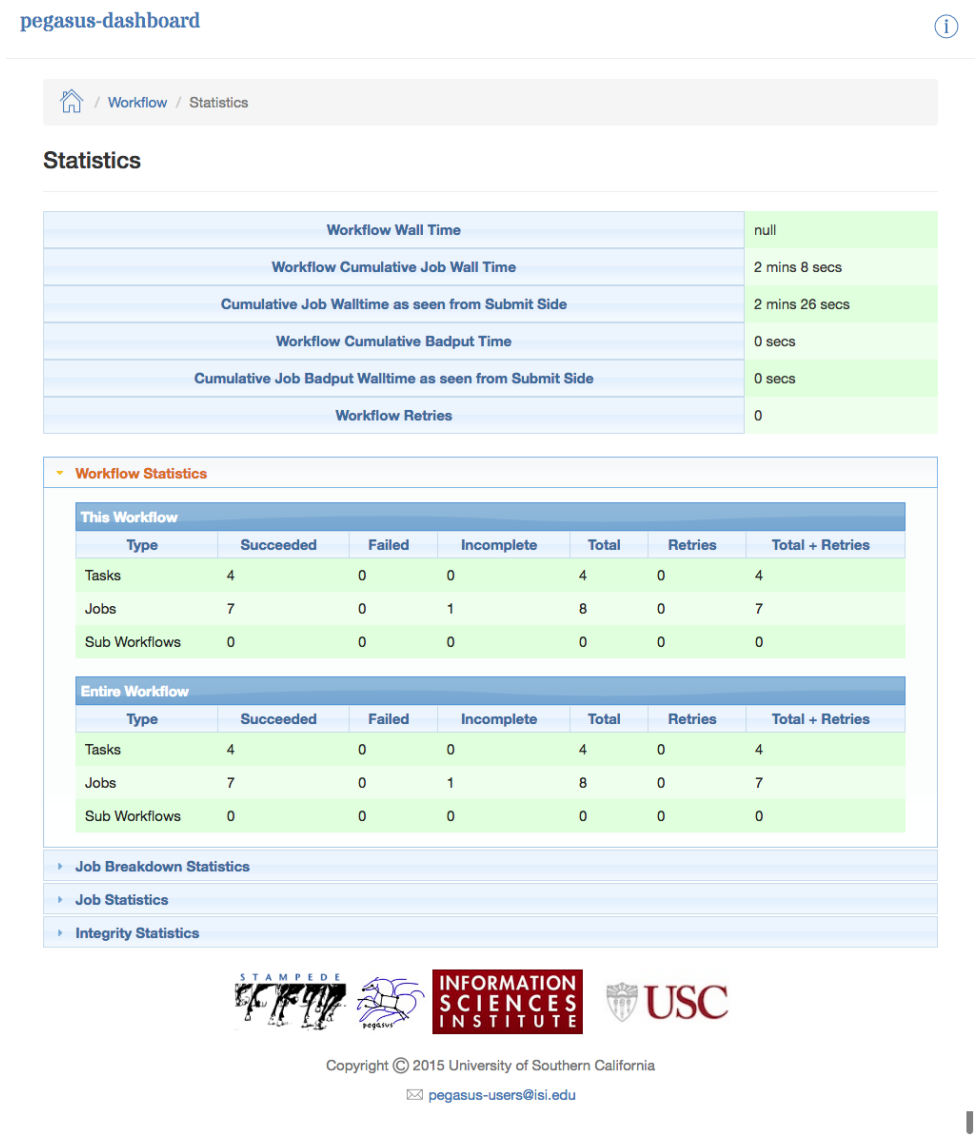

Figura 3.3: Página de estatísticas de *workflows* no Pegasus (DEELMAN et al., 2015).

3.4 Taverna

O Taverna também é caracterizado como um Sistema de Gerenciamento de Workflow Científico (SWfMS). O sistema apresenta as funcionalidades de orquestração de micros-serviços e de monitoramento. O *software* possui enfoque em aplicações de bioinformática, possuindo integração com outros serviços como: National Center for Biotechnology Information (NCBI), European Bioinformatics Institute (EMBL-EBI) e DNA Databank of Japan (DDBJ) (WOLSTENCROFT et al., 2013).

Alguns serviços utilizados nos *workflows* do Taverna possuem fácil acesso em um

ambiente criado para o compartilhamento desses serviços. O BioCatalogue[8] permite o registro de *web services* que integrem o Taverna. Além disso, os *workflows* criados, que utilizem os serviços dispostos pelo BioCatalogue, podem ser compartilhados em outro ambiente, o myExperiment[9]. Por meio desse *site*, cientistas podem detalhar como seus *workflows* funcionam e adicionar arquivos de configurações para que outros cientistas possam colaborar, replicar e utilizar seus resultados.

O Taverna teve suporte da Apache Incubator, entretanto foi descontinuado em 2020, como consta no *site* do projeto[10].

Um dos principais aspectos do Taverna é a disponibilidade de serviços padrões do sistema. Existem milhares de serviços disponíveis, principalmente no campo da bioinformática. O sistema possui uma interface gráfica de *desktop* que usa anotação semântica associada aos serviços. Os desenvolvedores podem adicionar novos serviços facilmente e podem carregar *workflows* existentes como um serviço, ou seja, é possível que componentes de novos *workflows* sejam também *workflows*.

A Figura 3.4 mostra uma das páginas da interface gráfica do Taverna.

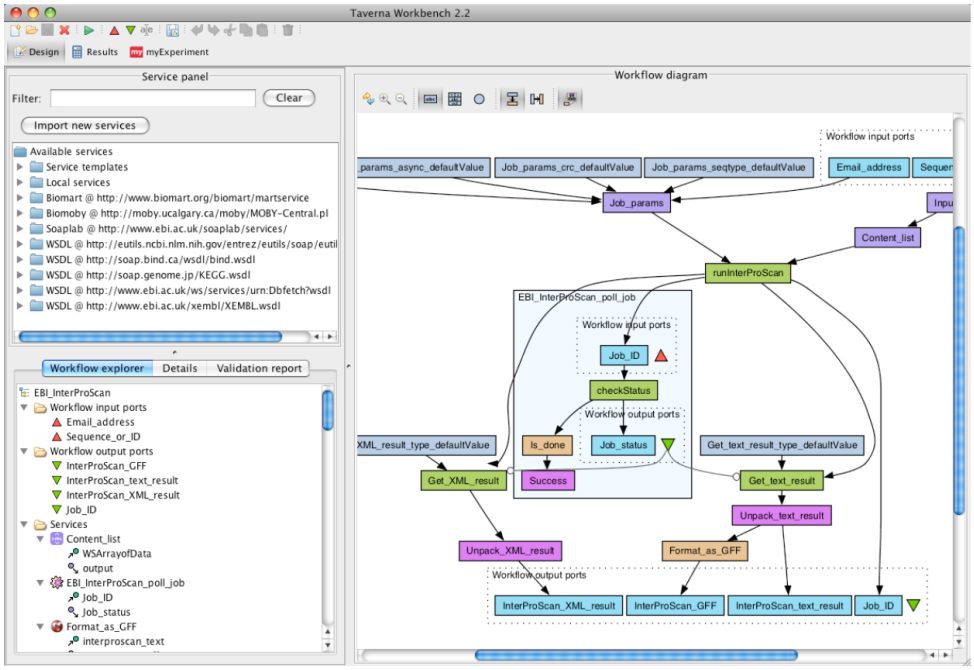

Figura 3.4: Página principal do Taverna (WOLSTENCROFT et al., 2013).

[8]https://www.biocatalogue.org
[9]https://www.myexperiment.org
[10]https://incubator.apache.org/projects/taverna.html

3.5 Kepler

Assim como o Pegasus, o Kepler, apresentado em Ludäscher et al. (2006) é um Sistema de Gerenciamento de Workflow Científico (SWfMS). Conforme destacado em Mandal et al. (2007), o Kepler possui uma interface intuitiva para design e execução, e seu paradigma de modelagem orientado ao ator o torna uma ferramenta muito versátil para design, prototipagem, execução e reutilização de fluxo de trabalho, tanto para engenheiros de fluxo de trabalho quanto para usuários finais.

Os atores presentes no Kepler facilitam a entrada e a organização de dados em diversos formatos, facilitando a integração entre serviços e disponibilizando recursos que permitem trabalhar com diversos formatos de arquivos, o que se demonstra muito útil quando é necessário analisar uma grande quantidade de dados, principalmente se esses dados tem origem em outras pesquisas que seguem determinados padrões de representação de dados. Um exemplo que o Kepler dispõe dentre suas funcionalidades é um ator de controle de Ecological Metadata Language (EML). Isso permite que o *software* importe e organize dados de uma maneira estruturada. O *software* ainda busca permitir que os próprios usuários desenvolvam atores que sejam integrados ao sistema.

O Kepler apresenta uma interface gráfica que utiliza técnicas de arrastar e soltar para definir *workflows*. Os atores são selecionados de dentro de uma biblioteca e são conectados para estabelecer as dependências. É utilizado um modelo de fluxo de dados onde os pacotes de dados são recebidos e enviados entre os atores diretamente. Essa abordagem permite execução paralela e utilização eficiente de recursos. Além disso, há suporte para execução distribuída e os atores podem ser distribuídos entre os recursos disponíveis.

O sistema foi pensado para suportar a integração de ferramentas externas. Para isso, são fornecidas uma API e uma arquitetura de *plugins*, o que permite estender as capacidades do Kepler.

O Kepler possui poucos recursos em termos de tolerância a falhas. Os usuário precisam tratar isso embutindo tratamento de exceções dentro dos *workflows* manualmente.

A Figura 3.5 apresenta uma tela da interface gráfica do Kepler.

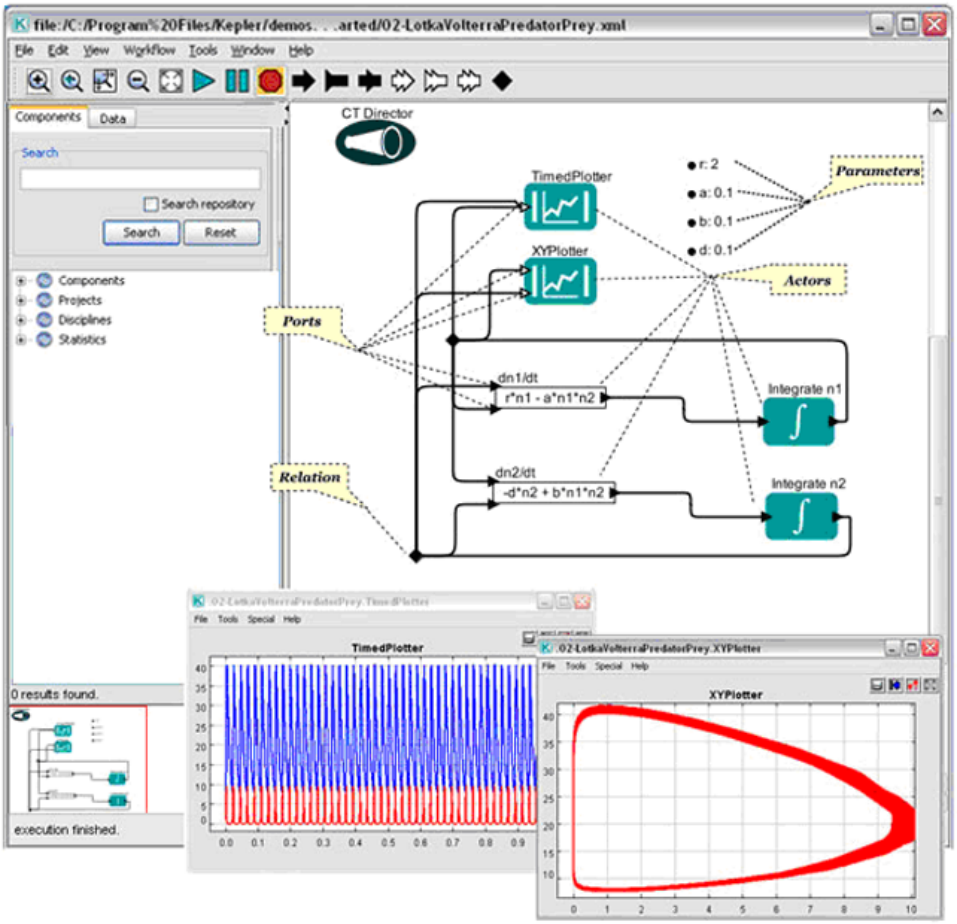

Figura 3.5: Página de construção de *workflows* do Kepler (LUDÄSCHER et al., 2006).

3.6 Triana

O Triana é um ambiente de aplicação de testes e experimentos de código aberto do projeto GridLab (ALLEN et al., 2003). O *software* foi desenvolvolvido para definir, processar, analisar, gerenciar, executar e monitorar *workflows*. A ferramenta permite que usuários componham *workflows* graficamente usando técnicas de arrastar e soltar componentes chamados de unidades em um espaço de trabalho. A conectividade das unidades é feita usando *links* de controle. O Triana tem a capacidade de distribuir partes de um *workflow* em máquinas remotas por meio de uma rede *peer-to-peer*. O programa fornece suporte a múltiplas linguagens, permitindo que diferentes editores de *workflow* sejam acoplados por *plugins* incluindo Web Services Flow Language (WSFL), DAGs, Business Process Execution Language (BPEL) e o formato de redes Petri (TAYLOR et al., 2007).

Um dos aspectos mais interessantes do Triana é sua interface gráfica. Ela apresenta ferramentas úteis de edição como agrupamento de entidades, recortar, copiar, colar, desfazer, editar entradas e saídas das tarefas, *zoom*, entradas opcionais e checagem de tipo.

O Triana apresenta muitos módulos padrões direcionados ao processamento de dados relacionados a análise de campo gravitacional, processamento de áudio, processamento de imagem e edição de texto.

Na Figura 3.6 é possível visualizar uma das páginas da interface gráfica do Triana.

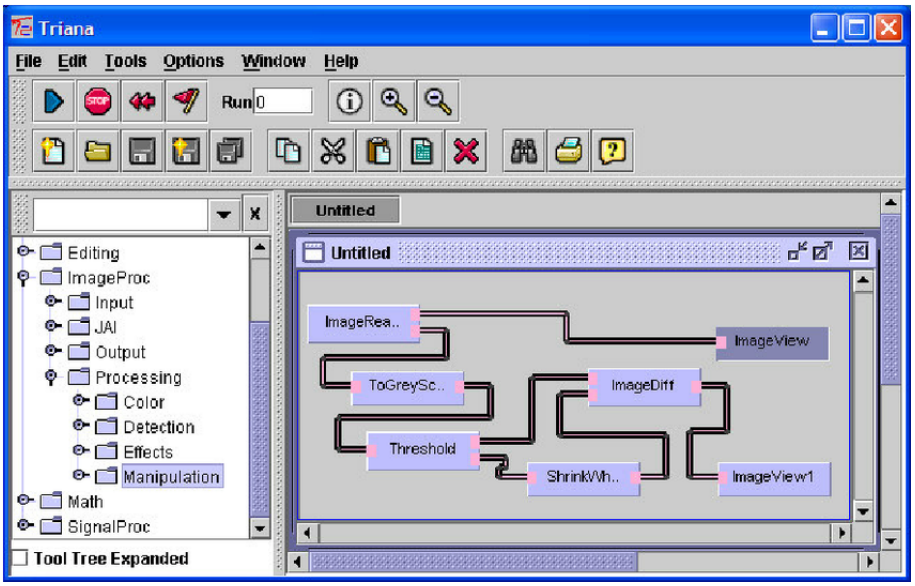

Figura 3.6: Página de visualização de *workflows* do Triana (TAYLOR et al., 2007).

3.7 Considerações finais

Os trabalhos apresentados ilustram as principais características presentes em WfMSs. É possível observar que há um padrão em determinadas funcionalidades, que estão presentes na maioria dos softwares descritos. Dessa maneira, é importante assegurar que essas características estejam presentes em um *framework* que se proponha a solucionar problemas similares ao que esses sistemas tentam também solucionar.

As interfaces gráficas estão presentes em todos os trabalhos, mostrando a importância de assegurar uma boa experiência ao usuário por meio de elementos que simplifiquem os conceitos teóricos de maneira visual e interativa.

4 Arquitetura do sistema

Este capítulo descreve a arquitetura do sistema proposto, que funciona como uma solução para o gerenciamento de *workflows*. Além disso, o capítulo apresenta, também, as decisões de implementação tomadas para viabilizar uma versão inicial do sistema. Todos os repositórios e códigos do sistema estão disponíveis publicamente na página do projeto no GitHub[11] e, atualmente, uma versão do sistema se encontra disponível em uma página *web*[12] hospedada nos servidores da Rede Integrada de Pesquisa em Alta Velocidade (RePesq) da UFJF. A Seção 4.1 apresenta uma síntese da proposta e do funcionamento do sistema, contextualizando termos que serão especificados em seções posteriores. A Seção 4.2 mostra como funciona o gerenciamento de módulos dentro do sistema. A Seção 4.3 apresenta os detalhes da integração de módulos de processamento à plataforma. A Seção 4.4 mostra como os módulos podem ser utilizados para compor um *workflow*, além de mostrar como se deu a implementação dos recursos. A Seção 4.5 apresenta detalhes sobre a API disponibilizada para acesso ao sistema. Por fim, a Seção 4.6 detalha a implementação e o funcionamento da interface gráfica.

4.1 Visão geral

O sistema proposto neste trabalho tem sua comunicação interna baseada em uma arquitetura de filas de mensagens, utilizando o protocolo AMQP. As filas são responsáveis por realizar a comunicação entre os serviços, que atuam como módulos da plataforma. Os módulos funcionam como unidades de processamento, podendo ser definidos por usuários e integrados ao sistema, ficando disponíveis para uso geral.

Apesar de não ser algo estritamente necessário, o sistema foi pensado para executar os módulos usando virtualização a nível de aplicação, ou seja, com o auxílio de um *container engine* que isole a aplicação em um contêiner, que por sua vez provê as dependências necessárias para a execução do módulo. A implementação proposta utiliza

[11] https://github.com/easytopic-project
[12] https://m2p.repesq.ufjf.br

o Docker como ferramenta para orquestração dos contêineres e algumas funcionalidades dependem de o módulo estar sendo executado de maneira virtualizada. Entretanto, um módulo que não utilize a ferramenta de virtualização ainda pode ser integrado ao sistema.

Diante de um ou mais módulos, um usuário pode definir um *workflow*, ou usar um dos *workflows* já definidos para fazer processamento de dados de seu interesse, enquanto o sistema gerencia a execução passo a passo.

Ao iniciar um processamento, a execução de um determinado *workflow* com os dados de entrada fornecidos pelo usuário é chamada de um *job* dentro do sistema. O usuário pode acompanhar a execução de seus *jobs* por API REST ou por interface gráfica, ferramentas que serão especificadas na Seção 4.5 e na Seção 4.6, respectivamente.

A Figura 4.1 apresenta um diagrama da arquitetura geral do sistema.

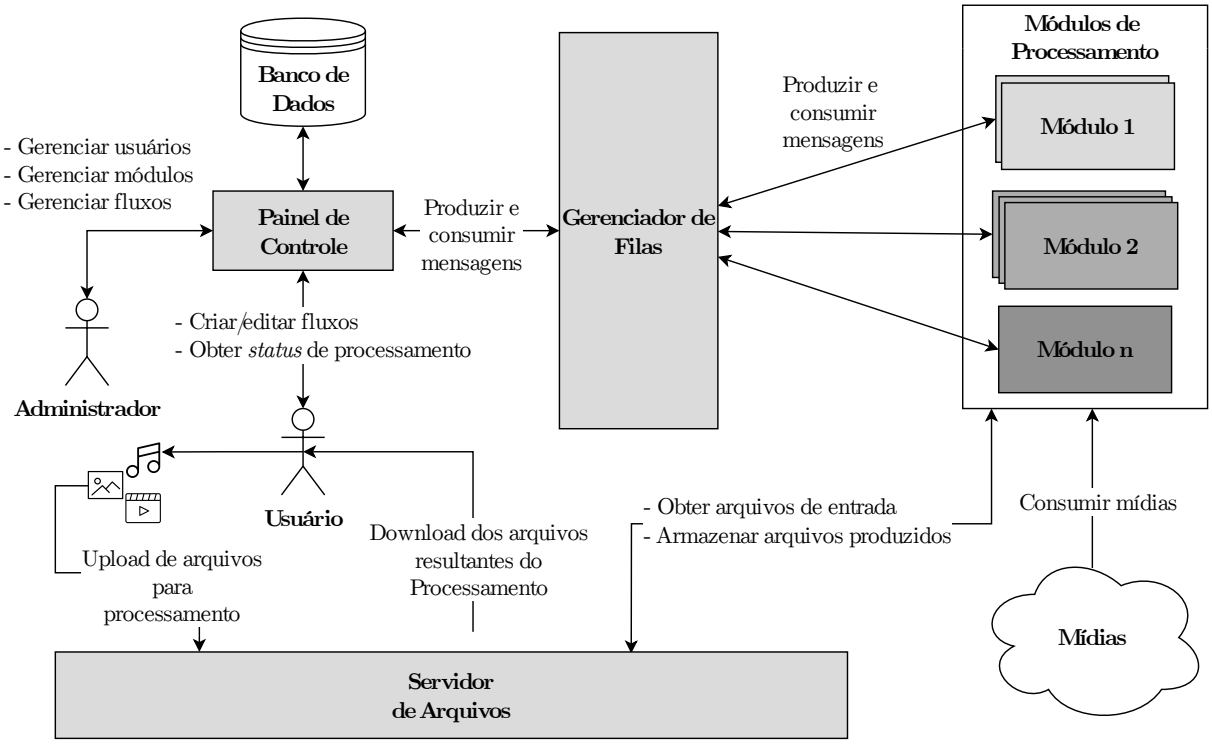

Figura 4.1: Arquitetura proposta para o sistema de gerenciamento de *workflows* M2P

O painel de controle do sistema foi implementado usando Node.js[13] e implementa interfaces de uso por meio de API REST e por interface gráfica. Esse painel atua gerenciando o fluxo do processamento de acordo com arquivos de configuração. Cada módulo do

[13]https://nodejs.org

sistema define uma fila de entrada e uma de saída no gerenciador de filas, e essas filas são coordenadas pelo painel de controle, que ao receber uma requisição de processamento, dispara mensagens nas filas de entrada dos módulos necessários. Simultaneamente, o painel de controle escuta as filas de saídas dos módulos, o que possibilita conhecimento sobre o fim de processamento de um módulo e o encaminhamento de uma mensagem de processamento para um módulo subsequente.

O sistema disponibiliza ao usuário um servidor de arquivos como solução para o *upload* de arquivos iniciais de processamento. Internamente os arquivos são tratados como *links* e são baixados pelos módulos que precisam utilizá-los. Arquivos intermediários resultantes dos processamentos também são armazenados nesse servidor de arquivos. Por isso, os módulos estão aptos a consumir mídias armazenadas em qualquer lugar, entretanto os resultados são todos armazenados no servidor de arquivos.

O banco de dados da aplicação foi implementado usando o MongoDB[14]. Como o painel de controle do sistema foi desenvolvido em Node.js, o armazenamento dos objetos de configuração e de resultados de processamento são feitos de maneira praticamente direta, já que o formato de representação e de troca de dados JSON é suportado e padronizado tanto no interpretador Node.js quanto no banco MongoDB.

4.2 Gerenciamento dos módulos

A aplicação fornece a um usuário administrador a possibilidade de gerenciar módulos que sejam executados na máquina que estiver atuando como servidor de todo o sistema. A ferramenta funciona disponibilizando rotas na API REST do sistema que permitem a manipulação de contêineres Docker na máquina. Para isso, a aplicação utiliza internamente a API do Docker Engine[15], abstraindo ao usuário parte da complexidade de construir os contêineres e gerenciá-los. Na Seção 4.5 e na Seção 4.6 serão apresentadas as rotas acessíveis por API REST e a interface gráfica que disponibilizam essa funcionalidade ao usuário.

[14]https://www.mongodb.com
[15]https://docs.docker.com/engine/api

4.3 Definição dos módulos

Um módulo, para o sistema, representa uma unidade que realiza um processamento pre-definido, podendo ser criado por usuários e disponibilizados publicamente. Um módulo se conecta ao sistema apenas pelo gerenciador de filas, de forma que deve escutar em uma fila de entrada, para que possa receber parâmetros de processamento e enviar os resultados em uma fila de saída. O módulo pode ser executado em um contêiner na máquina *host* da aplicação ou ser executado externamente, desde que escute as filas do gerenciador de filas da aplicação. Para isso, é necessário saber o endereço IP e a porta na qual o gerenciador de filas está sendo executado. Para facilitar e abstrair o conceito das filas para um criador de módulos, foi criada uma biblioteca em Python, para auxiliar na conexão com as filas. Por mais que essa biblioteca tenha sido desenvolvida como um facilitador para a criação de módulos que usem a linguagem Python, é possível desenvolver módulos em qualquer linguagem, estabelecendo uma conexão AMQP com o gerenciador de filas.

Cabe destacar que basta executar outras instâncias de um mesmo módulo para promover um escalonamento horizontal em termos de processamento. Isso ocorre devido ao funcionamento do Broker AMQP utilizado, em que mais de um consumidor pode escutar as mesmas filas. Dessa forma, o próprio gerenciador de filas cuida de balancear a entrega de mensagens entre módulos de processamento de acordo com a disponibilidade de cada um. De maneira simples, foi possível definir essa funcionalidade que pode aumentar significativamente a vazão de processamento de um *workflow*, ao mesmo tempo que abstrai a ideia para o usuário exigindo apenas uma execução paralela de diferentes instâncias do mesmo módulo.

Para ser integrado ao sistema e possibilitar seu uso em *workflows*, um módulo precisa especificar como deve receber os parâmetros para realizar seu processamento. Para isso, foi definido um arquivo de configurações, que um usuário deve criar para especificar uma interface de entrada para seu módulo. O padrão de um arquivo de configuração de módulo é apresentado na Figura 4.2.

```
{ "id": "sample-module",
  "name": "Sample Module",
  "description": "Sample module that receives an image and a
     text, and give everything back as output, but with the
     text backwords",
  "author": "Bruno Juca",
  "email": "brunojuca@ice.ufjf.br",
  "input_queue": "sample-module-in",
  "output_queue": "template-module-out",
  "input": [
    { "id": "input-image",
      "link": true,
      "type": "file",
      "required": true,
      "accept": ["video/mp4"] },
    { "id": "input-text",
      "link": false,
      "type": "text",
      "required": true }
  ],
  "output": [
    { "id": "echo-image",
      "link": true,
      "type": "file",
      "accept": ["video/mp4"] },
    { "id": "echo-text",
      "link": false,
      "type": "text" }
  ]
}
```

Figura 4.2: Arquivo JSON de configurações de um módulo no sistema M2P

4.4 Definição dos *workflows*

Um *workflow* é definido, dentro do sistema, como uma sequência de passos de processamento encadeados, que são executados pelos módulos disponíveis. É representado como um DAG, e é necessário definir todas as entradas e saídas dos passos que o compõem, na ordem desejada de execução. Um usuário precisa definir um arquivo de configurações, no qual todas essas informações são especificadas, ou utilizar a ferramenta de construção da interface gráfica. A Figura 4.3 apresenta um exemplo do arquivo.

```
{ "version": "1.0",
  "id": "ocr",
  "name": "OCR",
  "description": "Runs OCR on an image file",
  "input": [
    { "id": "image",
      "name": "Image",
      "description": "Input image with text",
      "type": "file",
      "required": true,
      "accept": ["image/*"] }
  ],
  "output": [
    { "id": "ocr",
      "from": "ocr-step:ocr",
      "type": "text",
      "name": "OCR Result",
      "description": "The result of the OCR processing" }
  ],
  "steps": [
    { "id": "ocr-step",
      "arguments": { "file": "image" },
      "output": ["ocr"] }
  ]
}
```

Figura 4.3: Arquivo JSON de configurações de um *workflow* no sistema M2P

No exemplo da Figura 4.3 foi especificado um *workflow* simples, de apenas um *step* de processamento. Um *step* representa a execução de um módulo, neste caso do módulo de reconhecimento ótico de caracteres (OCR, do inglês *Optical Character Recognition*) diante dos argumentos de entrada especificados. O vetor de *input* nesse caso contém apenas um objeto, que deve ser uma imagem. O vetor de *output* especifica quais os resultados de processamento que são relevantes para o usuário e de qual *step* ele vem, em um formato `"id-step-de-origem:output-do-step"` no campo `"from"`. Os *steps* são listados na ordem de execução desejada, e caso dois ou mais *steps* possam ser executados paralelamente, eles podem ser agregados em um *step* que tenha um campo `"type"` com valor `"aggregation"` e um campo de vetores `"steps"`, com seus próprios *steps* internos. O campo `"arguments"` dos *steps* faz um mapeamento das entradas que um módulo precisa com o nome daquele objeto dentro do código do módulo, que vão chegar pela mensagem de processamento na fila de entrada do módulo.

Quando um usuário deseja iniciar um fluxo de processamento, é necessário fornecer os *inputs* que foram especificados no arquivo de configuração do fluxo, seja por API ou por interface gráfica. O sistema recebe a solicitação e dispara os *steps* definidos, respeitando as ordens de execução. Esse início é realizado com a publicação de uma mensagem na fila de entrada do primeiro módulo de processamento, que publica uma mensagem com os resultados na sua fila de saída quando conclui o processamento. O sistema recebe a mensagem de conclusão e então dispara uma nova mensagem na fila do módulo definido no próximo *step*. Quando um *step* precisa usar como entrada a saída de um *step* anterior, o usuário deve informar como valor de um dos campos do objeto `"arguments"` da mesma forma que especifica o campo `"from"` das saídas em `"output"`.

4.5 API REST

A API REST do sistema foi desenvolvida utilizando a biblioteca Express para Node.js dentro do painel de controle do sistema. Ela fornece acesso aos principais recursos disponíveis aos usuários por meio de requisições HTTP. Foram definidas rotas e métodos HTTP que fazem a interface com o usuário. Um usuário pode fazer requisições diretamente à API, o que oferece flexibilidade para utilizar outras linguagens de programação

para integrar chamadas ao sistema.

As subseções seguintes apresentam as rotas e os métodos HTTP disponíveis. Alguns desses métodos possuem muitos parâmetros no corpo da requisição, que devem ser especificados em formato JSON. No código do sistema, foi usada uma biblioteca para auxiliar na geração da documentação, que foi integrada tanto à API quanto à interface gráfica. Pela API, a rota `/api/docs.json` retorna a documentação completa em formato JSON e pela interface gráfica a rota `/api/docs` mostra a documentação em tela. Essa documentação segue a especificação OpenAPI[16], padrão para descrição de APIs REST.

4.5.1 Rotas para gerenciamento de módulos

- GET `/modules`: retorna ao usuário os módulos disponíveis no sistema, com todas as suas informações em formato JSON;
- POST `/modules/stop`: para a execução de um módulo que esteja rodando dentro de um contêiner no servidor da aplicação. O identificador do módulo deve ser indicado no corpo da requisição;
- POST `/modules/start`: inicia a execução de um módulo que esteja rodando dentro de um contêiner no servidor da aplicação. O identificador do módulo deve ser indicado no corpo da requisição;
- POST `/modules/add`: adiciona um novo módulo ao sistema, recebendo no corpo da requisição as configurações do módulo e o caminho de um repositório GIT, contendo um arquivo Dockerfile para construção do contêiner do módulo no servidor.

4.5.2 Rotas para gerenciamento de *workflows*

- POST `/workflows/add`: adiciona um novo *workflow* ao sistema, recebendo suas configurações no corpo da requisição;
- GET `/workflows`: retorna os *workflows* disponíveis ao usuário em formato JSON.

[16] https://www.openapis.org

- POST `/workflows/{id}`: inicia o processamento de um *workflow* cujo identificador é especificado na URL. As entradas iniciais para o fluxo são fornecidas no corpo da requisição.

4.5.3 Rotas para gerenciamento de *jobs*

- GET `/jobs/{id}`: recupera informações sobre o estado do processamento do *job* especificado como parâmetro na URL;

- GET `/jobs`: recupera informações sobre o estado de todos os *jobs* executados pelo usuário.

4.6 Interface *web*

A interface *web* do sistema foi desenvolvida utilizando a biblioteca React, além de outras bibliotecas de componentes que auxiliaram na criação da aplicação, que consiste em uma Single Page Application (SPA), que roda em navegadores *web*. Uma SPA renderiza na tela os componentes de maneira dinâmica, sem a mudança de páginas tradicional em que há o recarregamento da página ao mudar de rotas. Isso permite uma fluidez maior na navegação e possibilita uma melhor experiência para o usuário. A aplicação também foi pensada para ser responsiva, se adaptando aos diferentes formatos de tela, incluindo telas menores como as de celulares. A interface se comunica com o sistema por meio da API REST disponibilizada.

Algumas das principais páginas da aplicação são: a página de seleção de *workflows*, a página de criação de *workflows*, a página de resultados de um *job* e a página de gerenciamento de módulos.

A página de seleção de *workflows*, apresentada na Figura 4.4 permite que o usuário selecione um dos *workflows* já definidos, e execute-os utilizando os arquivos iniciais fornecidos pelo próprio usuário em um formulário que é exibido na página seguinte à de seleção.

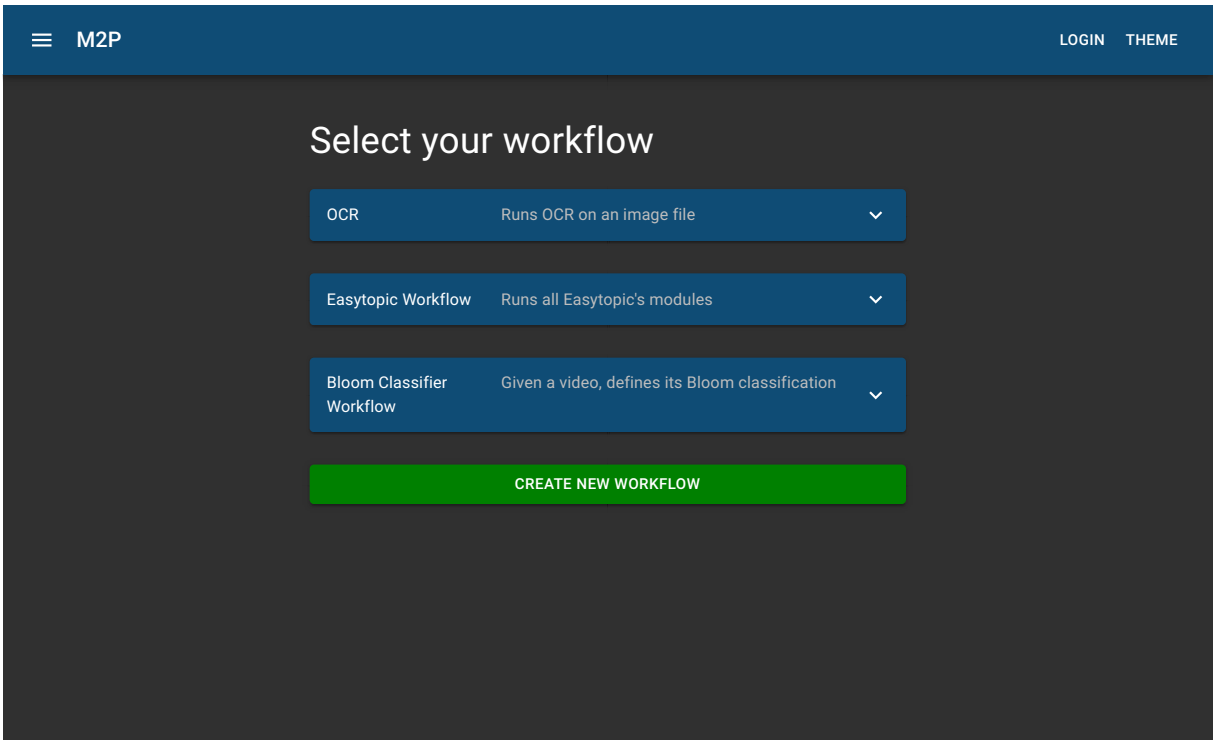

Figura 4.4: Página de seleção de *workflows* na interface *web* do M2P

A página de criação de *workflows* fornece ao usuário a possibilidade de criar graficamente um *workflow*, que segue uma representação interna igual à de um arquivo de configuração de *workflow*, em formato JSON. O usuário pode selecionar entradas iniciais e posteriormente adicionar *steps* ao seu *workflow*, que consiste na seleção de um dos módulos disponíveis e na definição de quais serão os arquivos de entrada dele, sejam eles provenientes de entradas iniciais, que são arquivos fornecidos pelo usuário, ou de saídas de processamento de outros módulos. A Figura 4.5 mostra o funcionamento da ferramenta, e é possível observar como um *workflow* se comporta como um grafo, mais especificamente como um DAG. Nesta página, também é possível importar um arquivo JSON com as configurações do *workflow* diretamente.

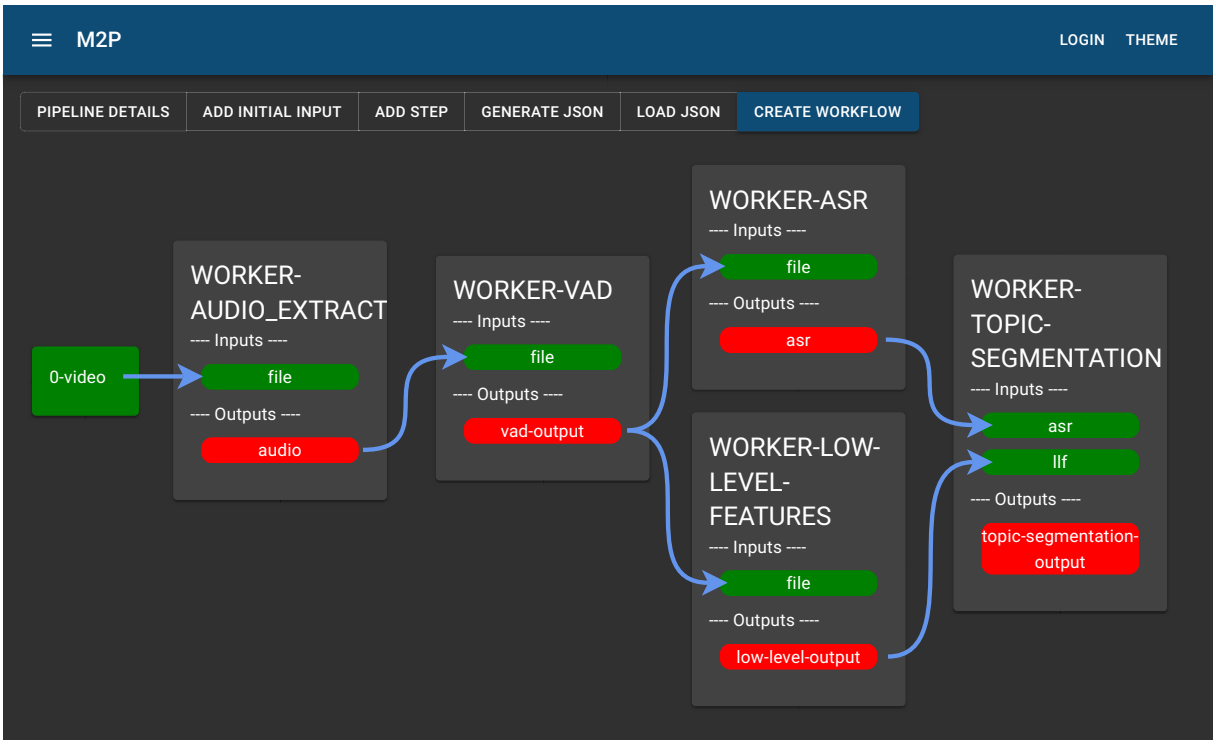

Figura 4.5: Página de criação de *workflows* na interface *web* do M2P

A página de resultados de um *job* exibe para o usuário as informações sobre o processamento realizado. Um menu de acesso lista todos os processamentos iniciados, que permite que o usuário navegue até essa página, que é individual para cada *job*. Nela estão presentes algumas estatísticas sobre o processamento e os resultados obtidos, com visualização prévia para alguns tipos de arquivos específicos como áudios, textos, imagens e vídeos. O *status* de processamento de cada passo é atualizado dinamicamente, mantendo o usuário informado sobre os estágios de execução. Além disso há opção de *download* para cada resultado intermediário. Essa página pode ser observada na Figura 4.6.

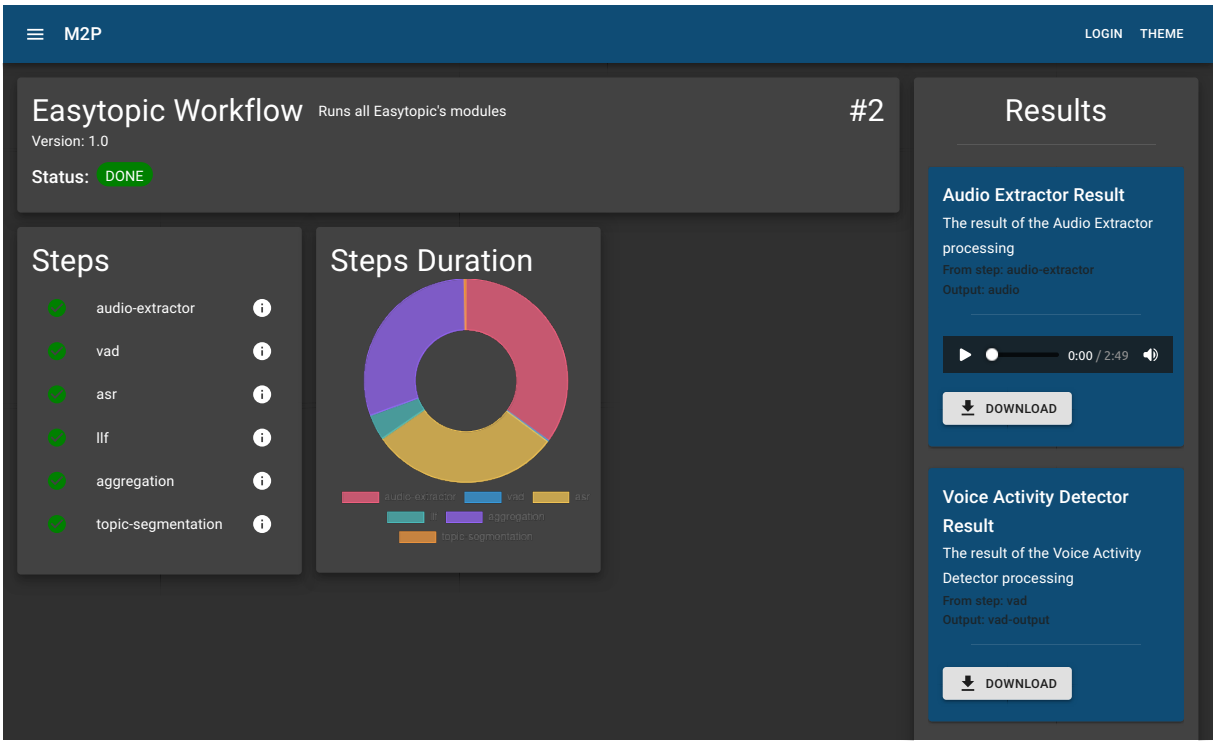

Figura 4.6: Página de resultados de um *job* na interface *web* do M2P

Por fim, a página de gerenciamento de módulos apresenta os módulos presentes na plataforma. Os módulos que são executados no servidor *host* do sistema podem ser administrados ativamente, enquanto módulos externos, executados em outras máquinas, têm apenas suas informações básicas apresentadas. Essa página é apresentada na Figura 4.7.

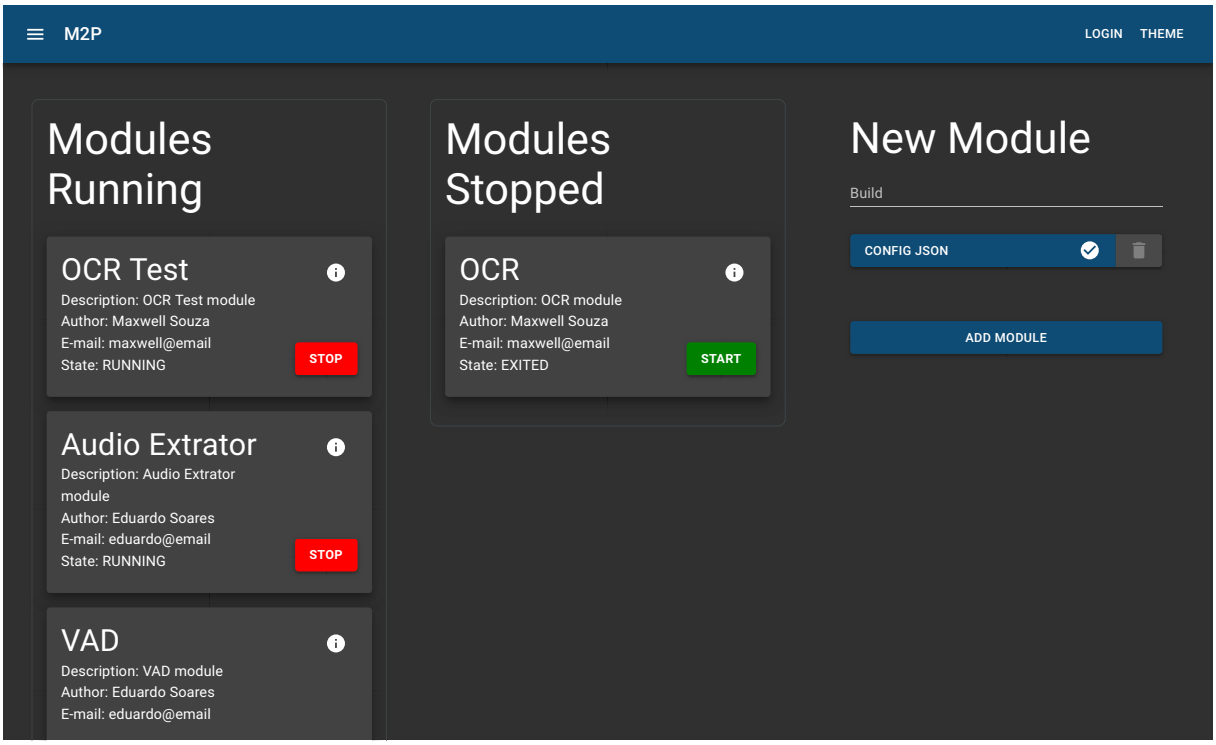

Figura 4.7: Página de gerenciamento de módulos na interface *web* do M2P

5 Testes

Com o intuito de validar o funcionamento do sistema e a sua aplicabilidade em problemas reais, foram realizados dois testes envolvendo projetos de pesquisa do LApIC. O primeiro refere-se ao trabalho em Barrére, Souza e Soares (2020), que busca realizar a segmentação automática de videoaulas em tópicos. O segundo envolve um projeto ainda em andamento, que busca apresentar automaticamente a classificação de uma videoaula de acordo com a taxonomia de Bloom, uma estrutura de organização hierárquica de objetivos educacionais. Os testes buscam validar o uso do sistema, produzindo resultados de processamento iguais àqueles obtidos sem o seu uso, sem prejudicar o desempenho de execução.

5.1 Easytopic

O Easytopic é um *framework* proposto em Barrére, Souza e Soares (2020) que realiza uma série de processamentos em um vídeo. O *workflow* do Easytopic recebe como parâmetro de entrada um vídeo e apresenta como resultado final sua segmentação em tópicos, com o intuito de facilitar a identificação de trechos relevantes para um usuário por meio de metadados. O *workflow* foi adaptado para funcionar na arquitetura do sistema M2P, realizando a configuração dos módulos internos e a configuração do arquivo de *workflow* do fluxo. O projeto Easytopic já apresenta uma arquitetura de filas, entretanto os módulos encaminham as mensagens de processamento diretamente uns para os outros, necessitando de conhecimento sobre o único fluxo possível e impedindo a reutilização dos módulos para outras tarefas.

O fluxo adaptado para o sistema M2P é apresentado no diagrama da Figura 5.1. Além disso, a Figura 4.3 mostra como o *workflow* foi configurado utilizando a interface gráfica.

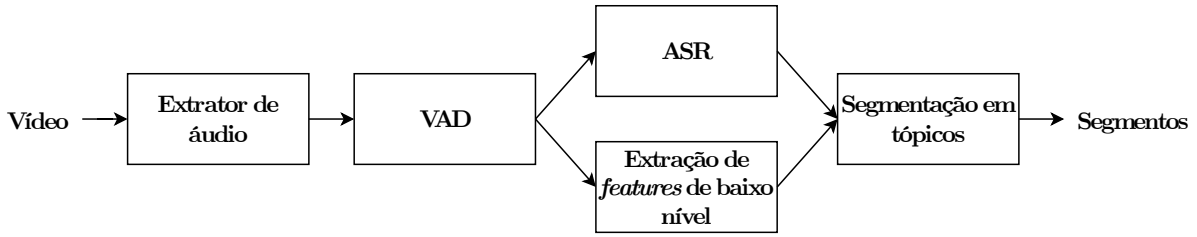

Figura 5.1: *Workflow* que mostra os módulos e as dependências do Easytopic

O *workflow* do Easytopic começa com um módulo que extrai o áudio de um vídeo. O áudio é encaminhado para um outro módulo de detecção de atividade vocal (VAD, do inglês *Voice Activity Detection*). Nesse momento do *workflow*, ocorre um fenômeno interessante para teste do sistema, em que a saída do VAD é encaminhada tanto para um módulo de reconhecimento automático de fala (ASR, do inglês *Automatic Speech Recognition*) quanto para um módulo de extração de *features* de baixo nível. Esses módulos são disparados simultaneamente pelo sistema, com a inserção de mensagens de processamento nas duas filas. Posteriormente, os dois módulos enviam o resultado de seus processamentos para suas respectivas filas de saída, o que ocorre, com alta probabilidade, em momentos distintos. O painel de controle cuida de disparar o módulo de segmentação em tópicos, que depende da saída dos dois módulos anteriores, somente quando os resultados de ambos estiverem disponíveis. Por fim, o resultado do módulo de segmentação em tópicos é disponibilizado.

O tratamento de agregação dos resultados dos módulos de ASR e de extração de *features* é solucionado no sistema com definições como a apresentada na configuração de um dos *steps* de execução do *workflow* do Easytopic, que pode ser observada na Figura 5.2. A solução funciona de forma recursiva, de forma que um *step* do tipo agregação pode ter outros *steps* dentro dele.

```
{ "id": "aggregation",
  "type": "aggregation",
  "steps": [
    { "id": "asr",
      "arguments": { "file": "vad:vad-output" },
      "output": ["asr"] },
    { "id": "llf",
      "arguments": { "file": "vad:vad-output" },
      "output": ["low-level-output"] }
  ]
}
```

Figura 5.2: *Step* de agregação no *workflow* do Easytopic

Os módulos foram executados utilizando contêineres Docker em execução na máquina *host* do sistema, e processaram os mesmos dados de teste da arquitetura do Easytopic sem erros de execução na maioria das vezes. O módulo de ASR apresentou problemas enventuais, mas nenhum relacionado ao sistema, e sim a erros internos de processamento. Dessa forma, o *workflow* do Easytopic pôde validar a integração de módulos ao sistema, a execução de um *workflow* de uso real, o gerenciamento de módulos da plataforma e a interface gráfica no processo de execução e obtenção dos resultados.

5.2 Classificador de taxonomia de Bloom

O *workflow* de classificação de taxonomia de Bloom utiliza uma videoaula como entrada, que passa por um primeiro módulo de ASR. Um dos intuitos desse teste seria realizar o reaproveitamento do módulo de ASR já existente na plataforma e configurado para o Easytopic, entretanto o módulo não apresentou funcionamento adequado para as videoaulas utilizadas como teste. Como já mencionado anteriormente, o módulo apresentou falhas internas desconhecidas. Uma vez que corrigir essas falhas não é o intuito deste trabalho, mas sim assegurar a coordenação da execução de módulos funcionais, para prosseguir com os testes deste *workflow*, foi utilizada uma base com o reconhecimento de fala das videoaulas já em mãos, e na prática o *workflow* foi executado desse ponto em diante.

O texto das videoaulas foi, então, encaminhado ao módulo de identificação de questões, que aponta quais trechos do vídeo contêm perguntas utilizadas para definir a classificação das videoaulas de acordo com a taxonomia de Bloom. Sequencialmente, os resultados foram passados para o módulo de classificação, que retornou os resultados esperados. A Figura 5.3 ilustra o *workflow*.

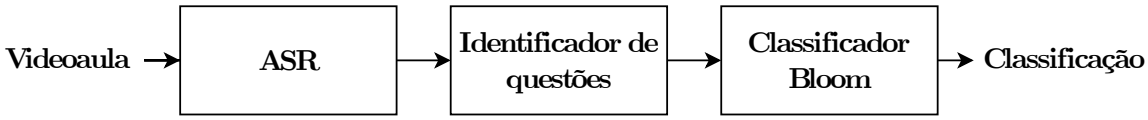

Figura 5.3: *Workflow* do classificador de taxonomia de Bloom

Diante desse teste, foi possível validar a execução de um *workflow* de uso real, a interface gráfica no processo de execução e obtenção de resultados e o funcionamento de módulos em ambientes distribuídos, já que os módulos de identificação de questões e de classificação foram executados em máquinas diferentes da máquina *host* de aplicação.

6 Conclusão

O trabalho realizado apresentou um sistema que funciona como um WfMS simples. Foram implementadas as principais funcionalidades que são capazes de definir, gerenciar e executar *workflows* por meio do sistema. O trabalho reuniu diferentes ferramentas que, quando colocadas em conjunto, facilitam o desenvolvimento de um sistema que lida com diferentes fluxos de processamento. Foi possível, ainda, definir uma API REST e uma interface gráfica para controle, que ainda podem ser expandidas para agregar funcionalidades adicionais.

O trabalho pôde dar continuidade a pesquisas anteriores, adicionando funcionalidades, definindo teoricamente o sistema como um WfMS e abordadando assuntos apontados como possibilidades de trabalhos futuros. Os testes realizados asseguraram o funcionamento esperado, mostrando a aplicabilidade do sistema em trabalhos realizados dentro do LApIC.

É possível apontar, como sugestão para trabalhos futuros, a implementação de recursos adicionais, que melhorem a experiência do usuário do sistema. Um sistema de recuperação de falhas robusto e com informações mais precisas sobre erros internos dos módulos é um exemplo. Além disso, não foram desenvolvidos muitos recursos de segurança além de autenticação por senha no sistema. Os módulos poderiam contar com recursos que assegurem o processamento proposto. O armazenamento dos resultados de processamento poderia ter integração com ambientes em nuvem, permitindo que o usuário informe o destino de todos os arquivos gerados, evitando assim uma sobrecarga do sistema. Apesar de já contar com recursos que viabilizam o processamento distribuído de dados, uma solução de replicação do próprio sistema poderia permitir o uso de múltiplas máquinas para compartilhamento de recursos usando apenas uma central de processamento, com outras máquinas atuando como nós e abstraindo ao usuário em qual máquina interna ao sistema um módulo está sendo executado estando dentro de um contêiner. Diante dos sistemas presentes nos trabalhos relacionados apresentados, pode ser interessante realizar uma comparação qualitativa mais aprofundada sobre suas diferenças.

Bibliografia

ALLEN, G.; GOODALE, T.; RADKE, T.; RUSSELL, M.; SEIDEL, E.; DAVIS, K.; DOLKAS, K. N.; DOULAMIS, N. D.; KIELMANN, T.; MERZKY, A.; NABRZYSKI, J.; PUKACKI, J.; SHALF, J.; TAYLOR, I. Enabling applications on the grid: A gridlab overview. *The International Journal of High Performance Computing Applications*, v. 17, n. 4, p. 449–466, 2003. Disponível em: <https://doi.org/10.1177/10943420030174008>.

BARKER, A.; HEMERT, J. van. Scientific workflow: A survey and research directions. In: WYRZYKOWSKI, R.; DONGARRA, J.; KARCZEWSKI, K.; WASNIEWSKI, J. (Ed.). *Parallel Processing and Applied Mathematics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 746–753. ISBN 978-3-540-68111-3.

BARRÉRE, E.; SOUZA, J. F. de; SOARES, E. R. Framework para segmentação temporal de vídeos educacionais. In: *Anais do XXXI Simpósio Brasileiro de Informática na Educação*. Porto Alegre, RS, Brasil: SBC, 2020. p. 972–981.

CUGOLA, G.; MARGARA, A. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 44, n. 3, p. 1–62, 2012.

CURRY, E. Message-Oriented Middleware. In: MAHMOUD, Q. H. (Ed.). *Middleware for Communications*. Chichester, England: John Wiley and Sons, 2004. cap. 1, p. 1–28. ISBN 978-0-470-86206-3.

DEELMAN, E.; GANNON, D.; SHIELDS, M.; TAYLOR, I. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, v. 25, n. 5, p. 528–540, 2009.

DEELMAN, E.; PETERKA, T.; ALTINTAS, I.; CAROTHERS, C. D.; DAM, K. K. van; MORELAND, K.; PARASHAR, M.; RAMAKRISHNAN, L.; TAUFER, M.; VETTER, J. The future of scientific workflows. *The International Journal of High Performance Computing Applications*, v. 32, n. 1, p. 159–175, 2018.

DEELMAN, E.; VAHI, K.; JUVE, G.; RYNGE, M.; CALLAGHAN, S.; MAECHLING, P. J.; MAYANI, R.; CHEN, W.; Ferreira da Silva, R.; LIVNY, M.; WENGER, K. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, v. 46, p. 17–35, 2015. ISSN 0167-739X.

GIL, Y.; RATNAKAR, V.; KIM, J.; GONZALEZ-CALERO, P.; GROTH, P.; MOODY, J.; DEELMAN, E. Wings: Intelligent workflow-based design of computational experiments. *IEEE Intelligent Systems*, v. 26, n. 1, p. 62–72, 2011.

HOLLINGSWORTH, D.; HAMPSHIRE, U. Workflow management coalition: The workflow reference model. *Document Number TC00-1003*, Document Status, v. 19, n. 16, p. 224, 1995.

JOHN, V.; LIU, X. A survey of distributed message broker queues. *CoRR*, abs/1704.00411, 2017.

JR., J. G.; DIAS, L. L.; SOARES, E. R.; BARRERE, E.; SOUZA, J. F. de. Framework for knowledge discovery in educational video repositories. *COMPUTING AND INFORMATICS*, v. 38, n. 6, p. 1375–1402, Feb. 2020.

KLUGE, M.; FRIEDEL, C. C. Watchdog – a workflow management system for the distributed analysis of large-scale experimental data. *BMC Bioinformatics*, BioMed Central, v. 19, n. 1, p. 1–13, 2018.

LUDÄSCHER, B.; ALTINTAS, I.; BERKLEY, C.; HIGGINS, D.; JAEGER, E.; JONES, M.; LEE, E. A.; TAO, J.; ZHAO, Y. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, v. 18, n. 10, p. 1039–1065, 2006. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.994>.

MANDAL, N.; DEELMAN, E.; MEHTA, G.; SU, M.-H.; VAHI, K. Integrating existing scientific workflow systems: The kepler/pegasus example. In: *Proceedings of the 2nd Workshop on Workflows in Support of Large-Scale Science*. New York, NY, USA: Association for Computing Machinery, 2007. (WORKS '07), p. 21–28. ISBN 9781595937155. Disponível em: <https://doi.org/10.1145/1273360.1273365>.

MCLENNAN, M.; KENNELL, R. Hubzero: A platform for dissemination and collaboration in computational science and engineering. *Computing in Science & Engineering*, v. 12, n. 2, p. 48–53, 2010.

MERKEL, D. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, Belltown Media, Houston, TX, v. 2014, n. 239, mar 2014. ISSN 1075-3583.

MITCHELL, R.; POTTIER, L.; JACOBS, S.; SILVA, R. F. d.; RYNGE, M.; VAHI, K.; DEELMAN, E. Exploration of workflow management systems emerging features from users perspectives. In: *2019 IEEE International Conference on Big Data (Big Data)*. [S.l.: s.n.], 2019. p. 4537–4544.

O'HARA, J. Toward a commodity enterprise middleware: Can amqp enable a new era in messaging middleware? a look inside standards-based messaging with amqp. *Queue*, Association for Computing Machinery, New York, NY, USA, v. 5, n. 4, p. 48–55, may 2007. ISSN 1542-7730.

ROSTANSKI, M.; GROCHLA, K.; SEMAN, A. Evaluation of highly available and fault-tolerant middleware clustered architectures using rabbitmq. In: *2014 Federated Conference on Computer Science and Information Systems*. [S.l.: s.n.], 2014. p. 879–884.

RUSINKIEWICZ, M.; SHETH, A. Specification and execution of transactional workflows. *Modern database systems*, ACM Press/Addison-Wesley Publishing Co., v. 1995, p. 592–620, 1995.

SCHEEPERS, M. J. Virtualization and containerization of application infrastructure: A comparison. In: *21st twente student conference on IT*. [S.l.: s.n.], 2014. v. 21.

TAYLOR, I.; SHIELDS, M.; WANG, I.; HARRISON, A. The triana workflow environment: Architecture and applications. In: _____. *Workflows for e-Science: Scientific Workflows for Grids*. London: Springer London, 2007. p. 320–339. ISBN 978-1-84628-757-2. Disponível em: <https://doi.org/10.1007/978-1-84628-757-2_20>.

WATADA, J.; ROY, A.; KADIKAR, R.; PHAM, H.; XU, B. Emerging trends, techniques and open issues of containerization: A review. *IEEE Access*, v. 7, p. 152443–152472, 2019.

WOLSTENCROFT, K.; HAINES, R.; FELLOWS, D.; WILLIAMS, A.; WITHERS, D.; OWEN, S.; SOILAND-REYES, S.; DUNLOP, I.; NENADIC, A.; FISHER, P.; BHAGAT, J.; BELHAJJAME, K.; BACALL, F.; HARDISTY, A.; HIDALGA, A. Nieva de la; VARGAS, M. P. B.; SUFI, S.; GOBLE, C. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research*, v. 41, n. W1, p. W557–W561, 05 2013. ISSN 0305-1048. Disponível em: <https://doi.org/10.1093/nar/gkt328>.

ZHENG, C.; THAIN, D. Integrating containers into workflows: A case study using makeflow, work queue, and docker. In: . New York, NY, USA: Association for Computing Machinery, 2015. (VTDC '15), p. 31–38. ISBN 9781450335737.