MAESTRO: Uma Abordagem para a Composição e Análise de *Workflows* Baseados em *Scripts* por Meio de Ontologias*

Luiz Gustavo Dias[1], Bruno Lopes[1], Daniel de Oliveira[1]

[1] Instituto de Computação – Universidade Federal Fluminense (UFF)

lgdias@id.uff.br, {bruno,danielcmo}@ic.uff.br

Abstract. *Specifying workflows that implement scientific experiments using scripts is challenging, mainly because multiple programs can implement each step of the experiment. Setting programs inadequately can cause inconsistencies due to incompatibility of formats, dependencies, etc. Furthermore, even if a script is well specified and adequately executed, analyzing the data produced without knowledge about the experiment's domain terms and how it was specified can become challenging. In this paper, we present the MAESTRO approach, which is based on ontologies and provenance data to help the composition and analysis of the workflow implemented as a script. The MAESTRO approach combines Experiment Lines and domain data concepts and uses reasoners to specify a script and support analytical queries. MAESTRO was evaluated through a feasibility study in the bioinformatics domain, and the results were promising.*

Resumo. *Especificar workflows que implementam experimentos científicos por meio de scripts é uma tarefa desafiadora principalmente porque cada etapa do experimento pode ser implementada por múltiplos programas. Uma escolha inadequada de programas pode causar inconsistências devido a incompatibilidade de formatos, dependências, etc. Ademais, mesmo que um script seja bem especificado e devidamente executado, analisar os dados produzidos sem que tenhamos conhecimento acerca dos termos do domínio do experimento e como ele foi especificado pode se tornar um desafio. No presente artigo apresentamos a abordagem MAESTRO que é baseada em ontologias e dados proveniência para auxiliar a composição e análise do workflow implementado como script. A abordagem MAESTRO combina o conceito de Linha de Experimento e dados de domínio, e utiliza reasoners para especificar um script e apoiar consultas analíticas. A MAESTRO foi avaliada por meio de um estudo de viabilidade na área de bioinformática e os resultados se mostraram promissores.*

1. Introdução

Muitos experimentos em Ciência Computacional e Engenharia (CSE, de *Computer Science and Engineering*) são baseados em simulações computacionais de larga escala, que frequentemente requerem recursos de Computação de Alto Desempenho (HPC, de *High-Performance Computing*) para produzir resultados em um tempo aceitável [Gil et al. 2007]. Os *Workflows* Científicos (a partir de agora referidos apenas como "*workflows*") são uma abstração capaz de representar esses experimentos. Os *workflows* de CSE são intensivos na produção de dados

*O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. Os autores gostariam ainda de agradecer ao CNPq (*grant* 311898/2021-1) e FAPERJ (*grant* E-26/202.806/2019) pelo apoio financeiro.

e podem ser definidos como um conjunto de transformações de dados e suas dependências. Eles são tradicionalmente especificados e executados usando Sistemas de Gerência de *Workflows* (SGWfs), que oferecem uma gama de recursos para auxiliar na composição, execução e monitoramento (*e.g.*, Pegasus [Deelman et al. 2005]). No entanto, nos últimos anos, tem sido observado um aumento no uso de linguagens de *script* (*e.g.*, Python) para especificar e executar os *workflows* [Barba et al. 2021]. Esse aumento se deve, em parte, à disponibilidade de bibliotecas que permitem a execução paralela dos *scripts* em ambientes HPC, como o Dask [Crist 2016].

Embora um *script* ofereça maior flexibilidade para o usuário especificar o *workflow* (com estruturas condicionais, de repetição, *etc*.) [Wang and Peng 2019], ele representa apenas um *workflow* isolado e não necessariamente o experimento como um todo. Experimentos científicos podem envolver a composição, execução e análise de múltiplos *workflows*, nos quais o usuário explora diferentes transformações de dados, programas e parâmetros até que seja possível confirmar ou refutar sua hipótese científica. Portanto, a representação de um experimento requer um nível de abstração mais elevado do que o oferecido por um único *script* [Dias et al. 2020a, Dias et al. 2019], a fim de possibilitar a representação de todas as escolhas possíveis que o usuário pode fazer. Dessa forma, torna-se imperativa a possibilidade de representar o experimento em diferentes níveis de abstração, permitindo que o usuário derive *workflows* concretos, ou seja, os *scripts* que possam ser executados, a partir de uma série de escolhas de transformações de dados, parâmetros, *etc*. Os *workflows* concretos são então executados, e a execução de cada programa pode gerar n tarefas que podem ser executadas em paralelo (*e.g.*, usando a biblioteca Dask), consumindo um subconjunto dos dados. Além disso, é fundamental que essas escolhas sejam guiadas por termos específicos do domínio do experimento, *e.g.*, biologia computacional.

Tomemos como exemplo o experimento simplificado ilustrado na Figura 1, da área de bioinformática com seus múltiplos níveis de abstração. O experimento consiste em três transformações de dados: A, B e C. Cada transformação está relacionada a um conceito específico do domínio do experimento, *e.g.*, *Alinhamento Múltiplo de Sequências*. Além disso, essas transformações podem ser implementadas por mais de um programa. Por exemplo, a transformação A pode ser implementada pelos programas A_1 e A_2. Algumas transformações também podem ser opcionais, como no caso da transformação B (representada com uma linha pontilhada). O usuário tem a liberdade de escolher quais programas deseja executar para implementar cada transformação, resultando em uma representação concreta do *workflow*, *i.e.*, um *script* que invoca os programas A_1, B_1 e C_1. Essa representação é então executada pelo interpretador e os programas podem ser executados em paralelo em recursos computacionais, como as tarefas A_{11}, A_{12} e A_{13}.

Diversas abordagens já oferecem apoio para a composição e análise de experimentos científicos em múltiplos níveis de abstração, porém a maioria delas é baseada no uso de SGWfs [Gil et al. 2010, Gil 2013, Marinho et al. 2017, Lamprecht et al. 2021] ou convertem um *script* em uma especificação compatível com um SGWf [Baranowski et al. 2012, Carvalho et al. 2017]. Mesmo quando as abordagens permitem o uso de níveis de abstração para especificar *workflows* implementados como *scripts* [Ristov et al. 2021, Filgueira et al. 2020], ainda há lacunas no que diz respeito à análise dos resultados do experimento. Durante a execução do *script* é necessário coletar os dados de proveniência [Freire et al. 2008] para permitir a análise, *a posteriori* ou em tempo de execução, e garantir a reprodutibilidade. Essa captura de dados já é realizada por bibliotecas como

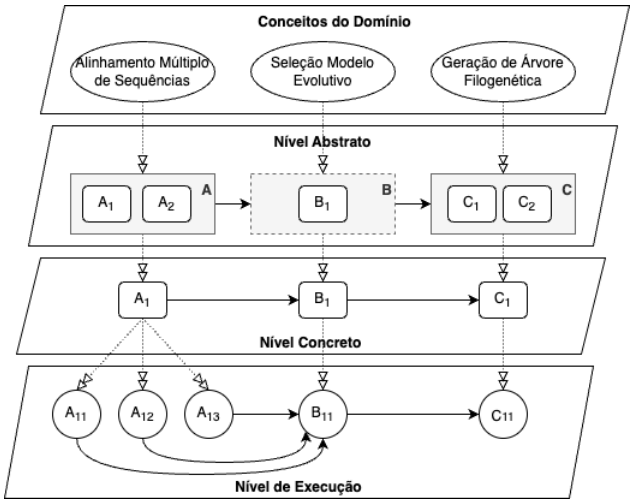

Figura 1. Representação do experimento em múltiplos níveis de abstração.

a Df-Analyzer [Silva et al. 2020], porém os dados de proveniência estão desconectados dos demais níveis de abstração do experimento e não estão associados aos termos do domínio.

Portanto, o presente artigo propõe a associação de uma ontologia [Guarino 1997] e dados de proveniência à representação abstrata do experimento por meio da abordagem MAESTRO, visando apoiar a composição e a análise de experimentos implementados como *scripts*. A MAESTRO utiliza uma ontologia que modela o conceito de Linha de Experimento [Marinho et al. 2017] para representar experimentos científicos em múltiplos níveis de abstração, abrangendo os principais conceitos e axiomas relacionados. Os usuários têm a possibilidade de especializar a ontologia por meio da associação com ontologias de domínio existentes, permitindo a utilização de termos específicos de um determinado domínio [Babalou and König-Ries 2020]. A partir de inferência nessa ontologia, é possível derivar os *scripts* por meio de uma API. Os *scripts* derivados são automaticamente instrumentados para que os dados de proveniência sejam automaticamente capturados usando a biblioteca Df-Analyzer. Ao final da execução dos *scripts*, o usuário pode consultar uma base que integra os dados de proveniência e os conceitos da ontologia. A MAESTRO foi avaliada por meio de um estudo de viabilidade com um experimento de geração de árvores filogenéticas no domínio da bioinformática [Dias et al. 2020b] e os resultados obtidos foram promissores, visto que a partir de uma única especificação abstrata foi possível gerar 30 *scripts* diferentes. Este artigo está organizado em quatro seções, além da Introdução. A Seção 2 apresenta o referencial teórico e discute trabalhos relacionados. A Seção 3 detalha a abordagem MAESTRO. A Seção 4 descreve o estudo de viabilidade realizado. Por fim, a Seção 5 conclui o artigo.

2. Referencial Teórico e Trabalhos Relacionados

2.1. Linhas de Experimento

Uma Linha de Experimento (LE) [Marinho et al. 2017] é uma abstração que permite representar as transformações de dados de um experimento científico em um nível alto de abstração. A LE é fundamentada em três conceitos principais: (i) Variabilidade, (ii) Opcionalidade e (iii) Derivação. A *Variabilidade* está relacionada às diferentes formas pelas quais uma transformação de dados pode ser implementada. Uma transformação é considerada um *Ponto de Variação* quando pode ser implementada por dois ou mais programas

diferentes. Caso contrário, a transformação é classificada como *Invariante*. Na Figura 1, a transformação A e C são pontos de variação, enquanto B é invariante. A *Opcionalidade* refere-se à característica de uma transformação de dados que pode estar presente ou ausente nos *scripts* criados a partir da representação abstrata. Na Figura 1, a transformação B é opcional, enquanto A e C são obrigatórias. Por fim, a *Derivação* é o processo de criar *scripts* a partir de uma definição abstrata do experimento e das escolhas feitas pelos usuários nos pontos de variação e opcionalidades. Além disso, a LE se baseia em uma representação algébrica, *i.e.*, cada transformação de dados consome uma relação e produz uma relação. Cada uma das relações é composta por múltiplos atributos que estão associados aos parâmetros de entrada e saída de cada programa do *script*.

Embora uma LE seja capaz de derivar diversos *workflows* implementados como *scripts*, na abstração original de LEs, não há nenhuma associação das transformações de dados com dados de proveniência, metadados ou terminologia específica do domínio do experimento, como os métodos ou operações de um domínio aos quais as transformações de dados da LE estão associadas. Por exemplo, a transformação A na Figura 1 está relacionada ao conceito de *Alinhamento Múltiplo de Sequências* da bioinformática, mas essa associação não era prevista na proposta da LE. Apenas adicionar um conjunto de metadados e/ou uma descrição textual nas transformações no nível abstrato poderia fornecer mais informações sobre a LE. No entanto, o uso de texto livre limita as buscas e resulta na falta de padronização dos termos, uma vez que cada usuário pode descrever uma transformação de maneiras diferentes. Além disso, metadados sem navegação ou inferência dificultam as consultas e análises *a posteriori*. Uma maneira de contornar as limitações supracitadas é por meio do acoplamento do conceito de LEs com ontologias, que são capazes de fornecer uma estrutura para a modelagem de conhecimento. O acoplamento com ontologias permite associar metadados, terminologia do domínio e informações importantes às transformações de dados da LE. A seguir, apresentamos a uma breve descrição da ontologia utilizada nesse contexto.

2.2. OntoExpLine: uma Ontologia para Representação de Linhas de Experimento

A *OntoExpLine* [Dias et al. 2020a] é uma ontologia que se baseia no conceito de LE e tem como objetivo auxiliar o processo de composição e análise de experimentos científicos que são executados por meio de *scripts*. Essa ontologia foi projetada para combinar os conceitos das LEs com outras ontologias existentes, incluindo: (i) a Ontologia ProvONE (`https://shre.ink/Q110`), que representa os conceitos de *workflows* e proveniência, (ii) o *Dublin Core Metadata Initiative* (DCMI) e seus metadados associados, e (iii) uma ontologia para representar conceitos de um domínio ou tarefas de um domínio. No contexto da avaliação realizada neste artigo, foi utilizada a ontologia EDAM (`http://edamontology.org/`), que é uma ontologia utilizada para representar análise de dados no domínio da bioinformática. A integração de múltiplas ontologias foi validada com o CoMerger [Babalou and König-Ries 2020]. As subontologias que compõem a *OntoExpLine* podem ser classificadas como *Mandatórias* e *Opcionais*, e um fragmento de sua estrutura é apresentado na Figura 2.

A subontologia ProvONE desempenha um papel mandatório na *OntoExpLine* ao definir os conceitos relacionados aos *workflows* (representados como *scripts*) que são gerados durante o processo de derivação. Essa subontologia consiste em 15 classes e 23 propriedades. Os conceitos do ProvONE são utilizados para representar *datasets*, execuções do *script*, e programas que podem ser invocados pelos *scripts*. Além disso, foram modeladas restrições na ProvONE para representar a compatibilidade de parâmetros de programas que podem ser

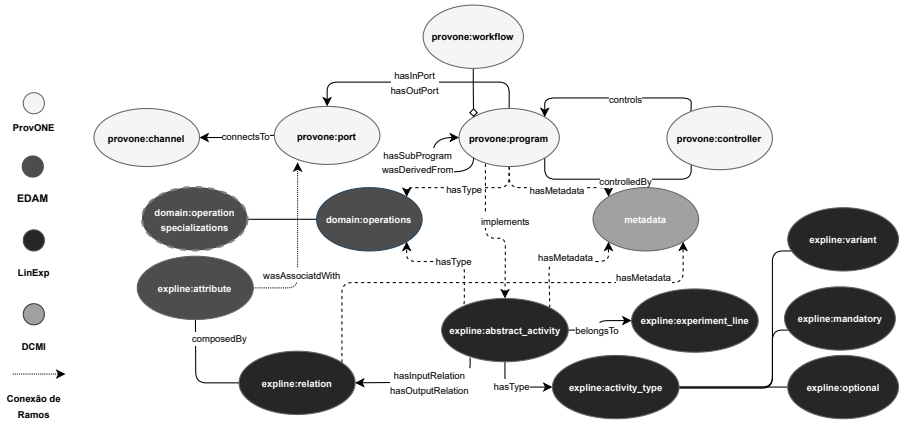

Figura 2. Fragmento da estrutura da *OntoExpLine*.

utilizados no *script*. Essa compatibilidade é fundamental para o processo de derivação por meio da ontologia. Além disso, os trechos de código utilizados para executar um programa ou função específica também são representados na ontologia.

Por outro lado, a subontologia EDAM é opcional na *OntoExpLine*. Isso significa que ela pode ser substituída por outras ontologias que representem análise de dados em domínio, contanto que essas ontologias contenham conceitos que representem operações/termos relevantes para o domínio do experimento. Essas operações estão associadas às transformações presentes na LE e, consequentemente, aos programas/funções que compõem o *script* após a derivação. Por sua vez, a subontologia DCMI é opcional e é utilizada para fornecer apoio na descoberta de recursos na *Web*, representando metadados. A DCMI é composta por 15 classes, permitindo a representação de autores, colaboradores, formatos, descrições e outros, relacionando-os a outros conceitos [Weibel and Koch 2000]. A documentação detalhada da *OntoExpLine* pode ser obtida em sua página no GitHub[1].

2.3. Biblioteca para Captura de Proveniência em *Scripts*

Existem diversas bibliotecas que permitem a captura de proveniência em *scripts*. Nesse trabalho, optamos por utilizar a biblioteca DfAnalyzer [Silva et al. 2020], que oferece recursos de monitoramento, depuração e análise de *script* em tempo de execução. Ela é capaz de capturar dados de proveniência presentes não apenas no *script*, mas também em arquivos, além de permitir o processamento de consultas durante a execução do *script*. A DfAnalyzer é agnóstica em relação à linguagem de programação, o que significa que os usuários podem utilizá-la em seus *scripts* independentemente da linguagem de programação. Para isso, a biblioteca disponibiliza um conjunto de serviços RESTful para a captura de proveniência. A DfAnalyzer difere das soluções existentes para análise de dados de proveniência em *scripts*, que geralmente estão vinculadas a uma linguagem de programação específica.

A DfAnalyzer captura dois tipos de dados de proveniência: (i) proveniência prospectiva e (ii) proveniência retrospectiva [Freire et al. 2008]. A proveniência prospectiva refere-se à especificação do experimento em um nível abstrato e aos *scripts* derivados. Já a proveniência retrospectiva representa os resultados gerados com a execução dos *scripts*, bem como dados do ambiente, *etc*. Para capturar esses dados de proveniência, o usuário precisa instrumentar o código, especificando no *script* quais transformações serão capturadas, quais

[1] `https://github.com/UFFeScience/OntoExpLine`

dados de entrada e saída são associados a cada transformação e em que ponto do código uma transformação é iniciada e concluída. Essa instrumentação é realizada por meio das chamadas à API RESTful, conforme exemplificado na Figura 3.

O *script* apresentado na Figura 3 representa um *workflow* com apenas uma transformação cujo objetivo é somar o valor de duas variáveis (linhas 11, 12 e 19). O restante do *script* é utilizado para definir as transformações e os dados de proveniência que serão capturados. Na linha 3, a transformação *ExecutarSoma* é definida. Cada transformação para a DfAnalyzer consome uma relação (parâmetros de entrada) e gera uma nova relação (parâmetros de saída), similarmente ao modelo seguido pela LE. Cada relação possui um conjunto de atributos com tipos definidos. A relação de entrada (*iExecutarSoma*) possui dois atributos (*PRIMEIRO_NUMERO* e *SEGUNDO_NUMERO*) e a relação de saída (*oExecutarSoma*) possui um atributo (*RESULTADO_SOMA*). Uma vez que a proveniência prospectiva é definida, o usuário precisa especificar os pontos de captura para a proveniência retrospectiva (*i.e.*, relativa à execução). Nas linhas 14, 15 e 16, o usuário define que uma tarefa chamada *ExecutarSoma* será executada, recebendo como entrada os valores das variáveis *NUMERO_1* e *NUMERO_2*, que são carregados nos atributos da relação *iExecutarSoma* definida anteriormente. Na linha 17, o usuário define o início da execução da transformação (ou seja, a soma propriamente dita) e, na linha 23, define o seu término. Isso permite que a DfAnalyzer capture o tempo de execução da transformação. Ao final da execução da transformação, a relação de saída é atualizada com o valor da soma dos dois números. É importante destacar que o processo de instrumentação do *script* não é uma etapa trivial e, atualmente, é realizado manualmente pelo usuário. A proposta deste artigo é que, por meio da abordagem apresentada, um *script* derivado de uma LE seja automaticamente instrumentado, permitindo a captura automática da proveniência pela DfAnalyzer.

```
1  dataflow_tag = "soma"
2  dflow = Dataflow(dataflow_tag)
3  transf1 = Transformation("ExecutarSoma")
4  transf1_input = Set("iExecutarSoma", SetType.INPUT,[Attribute("PRIMEIRO_NUMERO",
5      AttributeType.NUMERIC),Attribute("SEGUNDO_NUMERO", AttributeType.NUMERIC)])
6  transf1_output = Set("oExecutarSoma", SetType.OUTPUT,[Attribute("RESULTADO_SOMA", AttributeType.NUMERIC)])
7  transf1.set_sets([transf1_input, transf1_output])
8  dflow.add_transformation(transf1)
9  dflow.save()
10
11 NUMERO_1 = 5
12 NUMERO_2 = 1
13
14 task1 = Task(1, dataflow_tag, "ExecutarSoma")
15 task1_input= DataSet("iExecutarSoma", [Element([NUMERO_1, NUMERO_2])])
16 task1.add_dataset(task1_input)
17 task1.begin()
18
19 RESULTADO_SOMA = NUMERO_1 + NUMERO_2
20
21 task1_output= DataSet("oExecutarSoma", [Element([RESULTADO_SOMA])])
22 task1.add_dataset(task1_output)
23 task1.end()
```

Figura 3. Exemplo de instrumentação de um *script* com a DfAnalyzer.

2.4. Trabalhos Relacionados

A utilização de múltiplos níveis de abstração para representação de experimentos científicos não é uma proposta recente [Gannon et al. 2007]. No entanto, a grande maioria das abordagens é baseada na utilização de SGWfs. [Marinho et al. 2017] utilizam LEs para derivar *workflows* no SGWf VisTrails. A abordagem proposta por Marinho *et al.* permite a verificação de compatibilidade entre transformações de dados e os programas que as implementam, mas não possui conceitos do domínio associados, *i.e.*, o usuário deve conhe-

cer a modelagem da base de proveniência para realizar consultas analíticas diretamente em SQL. [Gil et al. 2010] e [Gil 2013] propõem uma abordagem baseada em ontologias para representação de *workflows* em um nível abstrato e seu posterior mapeamento para um *workflow* concreto no SGWfC Pegasus. Apesar de representar um avanço e permitir consultas associadas a termos do domínio, essas abordagens não associam dados da ontologia com dados de proveniência, limitando assim a capacidade analítica.

Por outro lado, a composição e análise de experimentos que são executados por meio de *scripts* ainda carecem de soluções. Apesar de soluções como as propostas por [Baranowski et al. 2012, Carvalho et al. 2017] permitirem que *scripts* se beneficiem das soluções implementadas em SGWfs (já que convertem *scripts* para linguagens de SGWfs), ainda apresentam limitações, uma vez que nem todas as estruturas de linguagem presentes em um *script* podem ser mapeadas para todos os SGWfs (*e.g.*, *loops*). Algumas abordagens já focam em prover níveis de abstração para representar um experimento que será executado via *scripts*. [Ristov et al. 2021] e [Filgueira et al. 2020] propõem arcabouços que permitem ao usuário representar seu experimento em um nível abstrato e mapeá-lo para um *script*. No entanto, diferentemente da `MAESTRO`, essas abordagens não provêm capacidades de análise dos resultados, pois não coletam dados de proveniência da execução, não permitindo assim uma integração da proveniência com a representação abstrata do experimento.

Apesar de especializar o modelo PROV, a subontologia ProvONE utilizada na *OntoExpLine* não foca na especificação de recursos (*e.g., scripts* ou programas). Dessa forma, abordagens como OWL-S [Martin et al. 2004], METEOR-S [Patil et al. 2004] ou WSMO [De Bruijn et al. 2005] adicionariam benefícios no processo de especificação no contexto da *OntoExpLine* assim como na derivação da especificação feita em LE, uma vez que tais abordagens visam descrever aspectos semânticos de recursos (*e.g.*, *scripts* e programas), permitindo sua invocação, interoperabilidade e descoberta.

3. Abordagem Proposta: `MAESTRO`

A `MAESTRO` é uma abordagem baseada no acoplamento de ontologias, LEs e dados de proveniência com o objetivo de oferecer apoio ao usuário nas etapas de composição, derivação e análise de experimentos científicos executados por meio de *scripts*. A arquitetura da `MAESTRO` é apresentada na Figura 4, e é composta de seis componentes principais: (i) API, (ii) Mecanismo de Derivação, (iii) Ontologia *OntoExpLine*, (iv) Mecanismo de Consulta, (v) Mecanismo de Proveniência e (vi) Base de Proveniência.

A API `MAESTRO` fornece detalhes sobre as operações que os usuários podem realizar para modelar seus experimentos, derivar *scripts* e realizar consultas que integram dados de proveniência e conceitos da ontologia. A API aceita requisições dos usuários em seus *endpoints* e retorna respostas de confirmação, utilizando códigos de resposta HTTP convencionais para indicar o sucesso ou falha de uma solicitação. Essas respostas podem também conter dados no formato JSON. Todas as solicitações para a API devem ser feitas através de uma conexão HTTP ou utilizando a biblioteca Python chamada `MAESTRO_lib_Python`. Para cada *endpoint*, o usuário pode fazer chamadas à API com seus próprios parâmetros. Por exemplo, a Figura 5 ilustra o uso da biblioteca `MAESTRO_lib_Python` para realizar as seguintes ações: selecionar uma ontologia (linha 1), escolher uma operação do domínio (linha 2), especificar parâmetros de entrada e saída da transformação (linhas 4 a 9), definir relações de entrada e saída da transformação (linhas 11 e 12), registrar um programa ou função (linha 13) e criar a transformação de dados chamada *validation*, que é implementada pelo programa/função

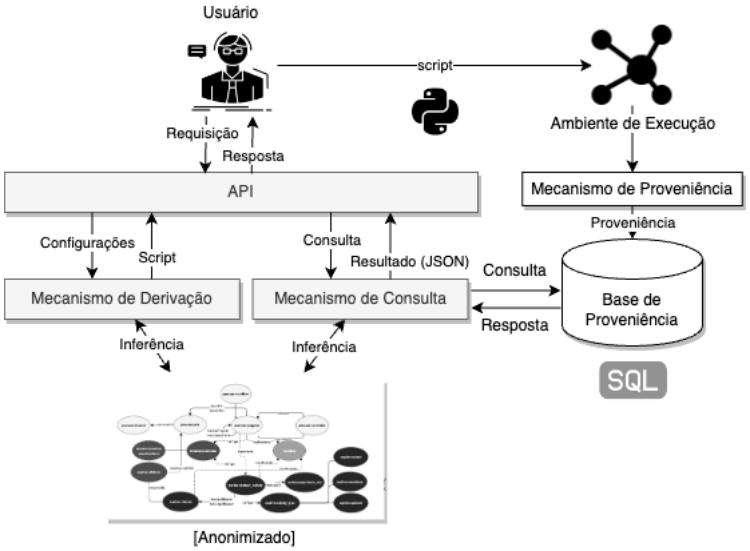

Figura 4. Arquitetura da MAESTRO.

remove_pipe e está associada à operação do domínio *Sequence_quality_control* da EDAM.

```
import maestro_lib_python
ontoexpline = get_ontology("ontologies/ontoexpline.owl").load()
op_validation = domainOperation(ontoexpline, "Sequencing_quality_control")
sequences_input = createAttribute(ontoexpline, "Input_Validation_att")
sequences_port = createPort(ontoexpline, "INPUT_SEQUENCE")
sequences_validated = createAttribute(ontoexpline, "Output_Validation_att")
sequences_validated_port = createPort(ontoexpline, "VALIDATED_SEQUENCE")
rel_input_validation = createRelation(ontoexpline, "Rel_Validation_In")
rel_output_validation = createRelation(ontoexpline, "Rel_Validation_Out")
associatePortAtt(sequences_validated_port, sequences_validated)
associateRelationAtt(rel_input_validation, [sequences_input])
associateRelationAtt(rel_output_validation, [sequences_validated])
remove_pipe = createProgram(ontoexpline, "remove_pipe", op_validation, "cgi-bin/arpa.py")
associateProgramPort(remove_pipe, [sequences_port], [sequences_validated_port])
aa = createActivity(ontoexpline, "validation", op_validation, [rel_input_validation],
    [rel_output_validation], False, [remove_pipe], True)
```

Figura 5. Exemplo de uso da API MAESTRO.

O *Mecanismo de Derivação* tem a responsabilidade de receber uma especificação de um *workflow* e realizar a inferência na *OntoExpLine* utilizando o *reasoner* Pellet para derivar um *script* executável. Ao considerar as escolhas de programas ou funções para implementar as transformações de dados na LE, o mecanismo verifica as dependências de dados entre esses programas ou funções. No caso de existir uma dependência entre dois programas p_i e p_j, o mecanismo verifica se a relação de entrada de p_j é um subconjunto da relação de saída de p_i, *i.e.*, se os parâmetros de entrada de p_j foram produzidos pelo programa p_i invocado imediatamente antes no *script*. Adicionalmente, o mecanismo adiciona automaticamente as marcações no *script* para que os dados de proveniência sejam capturados durante a sua execução pelo *Mecanismo de Proveniência*. Na versão atual, a MAESTRO realiza a instrumentação exclusivamente para a biblioteca DfAnalyzer. É importante ressaltar que todos os nomes de transformações, atributos e relações anotados para a DfAnalyzer (similares à Figura 3) são exatamente os mesmos que constam na *OntoExpLine*, o que permite a integração da *Base de Proveniência* com a ontologia. Finalmente, o *Mecanismo de Consulta* permite que o usuário especifique uma consulta capaz de integrar os dados de proveniência

capturados pelo mecanismo de proveniência e os termos da ontologia. O usuário submete essa consulta via API da MAESTRO. A MAESTRO se encontra em processo de disponibilização e poderá ser acessada em seu repositório[2].

4. Estudo de Viabilidade

Com o objetivo de realizar uma avaliação qualitativa da MAESTRO, conduzimos em conjunto com especialistas da área de bioinformática um estudo de viabilidade baseado em um experimento real de análise de árvores filogenéticas para o estudo de protozoários que infectam seres humanos. Em especial, utilizamos dados de diferentes espécies de parasita que causam a malária em humanos obtidos na base do NCBI[3]. Nesta seção, descreveremos como o experimento foi modelado na MAESTRO pelos especialistas e ilustraremos o processo de derivação e consulta sobre os dados de proveniência e os conceitos da ontologia.

O experimento escolhido como estudo de caso tem como objetivo construir árvores filogenéticas comparando centenas de genomas diferentes de parasitas que causam malária. A LE a ser modelada (Figura 6) é composta por cinco transformações de dados, sendo duas delas pontos de variação e uma opcional: (i) Formatação da Entrada, implementada por um *script* Python desenvolvido pelos especialistas, (ii) Alinhamento múltiplo de sequência, que pode ser implementada pelos programas MAFFT e ClustalW, (iii) Conversão de formato, implementada pelo programa Readseq e com execução opcional, (iv) Modelo evolutivo, implementada pelo programa ModelGenerator e (v) Geração da árvore, implementada pelos programas RAxML e MrBayes.

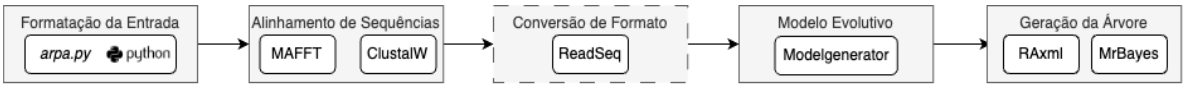

Figura 6. LE que representa o experimento de geração de árvores filogenéticas.

A partir da LE modelada, pretende-se derivar dois *scripts* que representam dois *workflows* concretos distintos, de acordo com as configurações apresentadas na Tabela 1. Na Derivação #1, a transformação de *Conversão de Formato* foi removida e os programas ClustalW e RAxML foram escolhidos para executar as transformações de *Alinhamento de Sequências* e *Geração da Árvore*, respectivamente. Já na Derivação #2, a transformação de *Conversão de Formato*, implementada pelo programa ReadSeq, foi mantida, e os programas MAFFT e MrBayes foram escolhidos para executar as transformações de *Alinhamento de Sequências* e *Geração da Árvore*, respectivamente. É importante ressaltar que, para que o MAFFT funcione em conjunto com o MrBayes, a etapa de conversão de formato é mandatória para garantir a compatibilidade. Essa informação se encontra representada na *OntoExpLine*.

De forma a especificar a LE, o primeiro passo foi definir quais são as transformações que farão parte da mesma, quais operações do domínio cada transformação está associada, que programas implementam quais operações, qual relação a transformação consome e qual relação ela produz. Um exemplo dessa composição é apresentado na Figura 5 onde a transformação *Validation* se encontra associada à operação do domínio *Sequencing_quality_control* e consome a relação de entrada que possui o atributo *INPUT_SEQUENCE*. Além disso, a transformação produz como saída a relação que possui

[2]https://github.com/UFFeScience/MAESTRO
[3]https://www.ncbi.nlm.nih.gov/assembly/GCF_000002765.3/

Tabela 1. Derivações para o experimento de geração de árvores filogenéticas.

Derivação #1		**Derivação #2**		
Programa/Função	**Parâmetro**	**Programa/Função**	**Parâmetro**	
RemovePipe	*-file_path*	RemovePipe	*-file_path*	
ClustalW	*Default*	MAFFT	*Default*	
~~ReadSeq~~	~~*Default*~~	ReadSeq	*Default*	
ModelGenerator	*Default*	ModelGenerator	*Default*	
RAxML	*Default*	MrBayes	ngen	100000
			nchains	4
			burnin	0
			nruns	2
			rates_ mrbayes	4

o atributo *VALIDATED_SEQUENCE* e é implementada pelo programa/função *arpa.py* que se encontra no diretório *cgi-bin*. Uma vez criada a transformação, ela é armazenada no objeto *aa*. O mesmo processo deve ser repetido para cada transformação da LE.

Uma vez que todas as transformações tenham sido definidas, a LE pode ser de fato criada. Inicialmente, o usuário define qual o conjunto de transformações fará parte da LE. No exemplo da Figura 7, as transformações *aa*, *aa2*, *aa3*, *aa4* e *aa5* foram selecionadas. Na linha 2 da Figura 7 as dependências entre as transformações são automaticamente inferidas na *OntoExpLine* usando o *reasoner* Pellet a partir das relações de entrada e saída de cada transformação. É importante ressaltar que esse tipo de inferência da estrutura acaba limitando os *scripts* que podem ser derivados, mas optamos por uma estrutura mais simples para avaliar a MAESTRO em um primeiro momento. Uma vez que a LE tenha sido criada, o usuário é capaz de derivar o *script* (linha 3), escolhendo os programas que implementam transformações que são pontos de variação. A derivação também é realizada por meio de inferência na ontologia, considerando compatibilidade entre os programas, dependências de dados, *etc*. No exemplo da Figura 7 o objeto *derivation* contém o *script* que foi gerado em Python. Finalmente, as chamadas para a DfAnalyzer são inseridas no *script* derivado previamente (linha 5). Para esse estudo de caso, o processo de derivação durou 0,382 segundos em média (30 derivações foram executadas).

```
1  absWf = [aa, aa2, aa3, aa4, aa5]
2  abs_wf = absWfDependencies(ontoexpline, absWf)
3  derivation = isValid(ontoexpline, abs_Wf,
4      [[aa2, clustalw], [aa5, mrbayes]], True)
5  createProvenanceCalls(ontoexpline, derivation,
6      [[aa2, clustalw], [aa5, mrbayes]])
```

Figura 7. Exemplo de uso da API MAESTRO para criação da LE com instrumentação para captura de proveniência com a DfAnalyzer.

No exemplo da Figura 7 podemos perceber que o usuário definiu explicitamente que o ClustalW e o MrBayes seriam os programas usados na derivação. Entretanto, nem sempre o usuário conhece todas as opções disponíveis. Por meio de uma consulta à MAESTRO, o usuário é capaz de buscar transformações equivalentes que tenham sido definidas na *OntoExpLine* conforme apresentado na Figura 8. A consulta por transformações equivalentes é

também realizada por meio de inferência sobre as propriedades atribuídas às instâncias de transformações de dados na *OntoExpLine*. No contexto da MAESTRO, as equivalências são transformações que possuem em suas relações de entrada e saída os atributos informados como referência. O mesmo raciocínio é aplicado para identificar programas que são equivalentes e/ou compatíveis.

```
ontoexpline = get_ontology("ontologies/ontoexpline.owl").load()
with ontoexpline:
    class Equivalence(ontoexpline.Entity):
        equivalent_to = [ontoexpline.Abstract_activity
            and (ontoexpline.hasInputRelation.some(ontoexpline.Relation
            and ontoexpline.composedBy.value(ontoexpline.alignment_att)))
            and (ontoexpline.hasOutputRelation.some(ontoexpline.Relation
            and ontoexpline.composedBy.value(converted_alignment_att_eq)))]
    ontoexpline.save(file="ontologies/ontoexpline.owl", format="rdfxml")
    close_world(Thing)
sync_reasoner(infer_property_values = True)
eq = ontoexpline.search(type = ontoexpline.Equivalence)
print(eq)
```

Figura 8. Inferência para identificação de transformações equivalentes.

Após a derivação e execução do *script*, o usuário pode submeter consultas que integram dados de proveniência capturados e conceitos da *OntoExpLine*. Por exemplo, suponha que o usuário queira identificar quais execuções de *scripts* possuem um programa associado à operação do domínio *Evolutive_model_generation* (que escolhe o melhor modelo de substituição de nucleotídeos ou aminoácidos) e que durante sua execução selecionou o modelo evolutivo JTT (que considera a substituição de aminoácidos representada como matrizes de taxa de substituição). Essa consulta é importante para os especialistas, pois o modelo JTT depende de várias suposições sobre os dados, que podem levar a erros e descartar uma grande quantidade de informações das sequências. Portanto, os usuários precisam avaliar se a aplicação do modelo JTT é válida ou não. Essa consulta envolve um conceito da ontologia (operação do domínio *Evolutive_model_generation*) e dados de proveniência (valor do atributo *evolutive_model* na base de proveniência). A Figura 9 ilustra o uso da API MAESTRO com o *endpoint search_by_domain_operation*, que retorna os dados dos *scripts* que executaram a operação do domínio *Evolutive_model_generation* e selecionaram o modelo JTT. A consulta da Figura 9 executou em 0,112 segundos em média.

```
ontoexpline = get_ontology("ontologies/ontoexpline.owl").load()
op_model = domainOperation(ontoexpline, "Evolutive_model_generation")
search_by_domain_operation(ontoexpline, op_model,
    parameters={"attribute": evolutiveModel_att, "port_value": "JTT"})
```

Figura 9. Consulta que envolve dados de proveniência e dados da ontologia.

O resultado da consulta supracitada é retornado em formato JSON, conforme apresentado na Figura 10. O arquivo JSON contém informações sobre a operação do domínio consultada, o *dataflow* (*Dataflow_id = 4*) que possui um programa que implementa essa operação e quais execuções desse *script* escolheram o modelo JTT (*Dts_execution_id: [14, 16, 18, 20, 22, 24]*). Caso o usuário não estivesse utilizando a abordagem MAESTRO, a conexão entre a operação do domínio e os dados de proveniência do *script* executado seria tácita, e o usuário teria que implementar um programa para realizar essa integração, o que pode não ser trivial. Essa integração automática entre a representação da LE e os dados de proveniência foi apontada pelos especialistas como a maior vantagem da MAESTRO.

```
{
    "Dataflows":{
        "Operation":"Evolutive_model_generation",
        "Dataflow_id": 4,
        "Attribute:": "model",
        "Value:" JTT",
        "Dts_execution_id:":[14, 16, 18, 20, 22, 24]
    }
}
```

Figura 10. Arquivo JSON com resultado de uma consulta.

5. Conclusões e Trabalhos Futuros

Representar experimentos científicos executados por meio de *scripts* usando diferentes níveis de abstração ainda é um desafio. Embora existam diversas soluções voltadas para SGWfs e algumas específicas para *scripts*, essas abordagens ainda não permitem consultas analíticas que integrem a representação abstrata do experimento com os dados de proveniência coletados durante a execução dos *scripts*. Neste artigo, apresentamos a abordagem `MAESTRO`, que combina ontologias e dados de proveniência para auxiliar na composição e análise de *workflows* implementados com *scripts*. A ontologia utilizada pela abordagem define conceitos LE, conjuntos de dados e proveniência, permitindo a derivação de *scripts* a partir de uma representação de mais alto nível com a captura de proveniência incorporada. Com a integração desses dados nos diferentes níveis de abstração, o usuário pode utilizar o conhecimento do domínio para realizar análises mais ricas dos dados de proveniência. A abordagem `MAESTRO` foi avaliada por meio de um estudo de viabilidade com especialistas da área de bioinformática, e a disponibilização de consultas que integram dados de proveniência e conhecimento do domínio foi considerada uma vantagem da abordagem proposta, além de ser uma *feature* que será usada no trabalho diário dos especialistas. Como trabalhos futuros, planejamos aplicar/estender a `MAESTRO` em outros domínios além da bioinformática.

Referências

[Babalou and König-Ries 2020] Babalou, S. and König-Ries, B. (2020). Towards Multiple Ontology Merging with CoMerger. In *Proc. of the ISWC 2020*, pages 59–64.

[Baranowski et al. 2012] Baranowski, M. et al. (2012). Constructing workflows from script applications. *Sci. Program.*, 20(4):359–377.

[Barba et al. 2021] Barba, L. A. et al. (2021). Scientific computing with python on high-performance heterogeneous systems. *Comput. Sci. Eng.*, 23(4):5–7.

[Carvalho et al. 2017] Carvalho, L. A. M. C. et al. (2017). NiW: Converting Notebooks into Workflows to Capture Dataflow and Provenance. In *Proc. of the K-CAP*, pages 12–16.

[Crist 2016] Crist, J. (2016). Dask & Numba: Simple libraries for optimizing scientific python code. In *IEEE BigData 2016*, pages 2342–2343.

[De Bruijn et al. 2005] De Bruijn, J., Bussler, C., Domingue, J., Fensel, D., Hepp, M., Kifer, M., König-Ries, B., Kopecky, J., Lara, R., Oren, E., et al. (2005). Web service modeling ontology (wsmo). *Interface*, 5(1):50.

[Deelman et al. 2005] Deelman, E. et al. (2005). Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237.

[Dias et al. 2020a] Dias, L., Lopes, B., and de Oliveira, D. (2020a). Ontoexpline: Rumo a uma ontologia para representação de linhas de experimento algébricas. In *Anais do XIV Brazilian e-Science Workshop*, pages 33–40, Porto Alegre, RS, Brasil. SBC.

[Dias et al. 2019] Dias, L. G., Lopes, B., and de Oliveira, D. (2019). Aplicação de ontologias de proveniência em workflows científicos: um mapeamento sistemático. In *Anais do XIII Brazilian e-Science Workshop*, Porto Alegre, RS, Brasil. SBC.

[Dias et al. 2020b] Dias, L. G., Mattoso, M., Lopes, B., and de Oliveira, D. (2020b). Experiencing dfanalyzer for runtime analysis of phylogenomic dataflows. In *Anais do XIII Simpósio Brasileiro de Bioinformática*, pages 105–116, Porto Alegre, RS, Brasil. SBC.

[Filgueira et al. 2020] Filgueira, R. et al. (2020). Dispel4py: An open-source python library for data-intensive seismology. In *EGU General Assembly Conf.*, page 6790.

[Freire et al. 2008] Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008). Provenance for computational tasks: A survey. *Comput. Sci. Eng.*, 10(3):11–21.

[Gannon et al. 2007] Gannon, D. et al. (2007). In *Workflows for e-Science, Scientific Workflows for Grids*, pages 1–8. Springer.

[Gil 2013] Gil, Y. (2013). Mapping semantic workflows to alternative workflow execution engines. In *2013 IEEE ICSC*, pages 377–382. IEEE Computer Society.

[Gil et al. 2007] Gil, Y. et al. (2007). On the black art of designing computational workflows. In *Proc.s of the WORKS*, page 53–62, New York, NY, USA.

[Gil et al. 2010] Gil, Y., Ratnakar, V., and Fritz, C. (2010). Assisting scientists with complex data analysis tasks through semantic workflows. In *AAAI Fall Symposium*. AAAI.

[Guarino 1997] Guarino, N. (1997). Understanding, building and using ontologies. *Int. J. Human-Computer Studies*, 46(2-3):293–310.

[Lamprecht et al. 2021] Lamprecht, A. et al. (2021). Perspectives on automated composition of workflows in the life sciences. *F1000Research*, 10:897.

[Marinho et al. 2017] Marinho, A. et al. (2017). Deriving scientific workflows from algebraic experiment lines: A practical approach. *FGCS*, 68:111–127.

[Martin et al. 2004] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., et al. (2004). Owl-s: Semantic markup for web services. *W3C member submission*, 22(4).

[Patil et al. 2004] Patil, A. A., Oundhakar, S. A., Sheth, A. P., and Verma, K. (2004). Meteor-s web service annotation framework. In *Proceedings of the 13th international conference on World Wide Web*, pages 553–562.

[Ristov et al. 2021] Ristov, S. et al. (2021). AFCL: An abstract function choreography language for serverless workflow specification. *FGCS*, 114:368–382.

[Silva et al. 2020] Silva, V. et al. (2020). Dfanalyzer: Runtime dataflow analysis tool for computational science and engineering applications. *SoftwareX*, 12:100592.

[Wang and Peng 2019] Wang, G. and Peng, B. (2019). Script of scripts: A pragmatic workflow system for daily computational research. *PLoS Comput. Biol.*, 15(2).

[Weibel and Koch 2000] Weibel, S. L. and Koch, T. (2000). The dublin core metadata initiative. *D-lib magazine*, 6(12):1082–9873.