# 2
# Preliminares

## 2.1.
## Estado da Arte

### 2.1.1.
### Workflows Científicos

As primeiras definições de *workflow* surgiram como uma forma de descrever processos realizados em escritório, que configuram uma sequência bem definida de tarefas que consomem documentos e são realizadas por pessoas ou sistemas de software. Em 1996, a WfMC (Workflow Management Coalition) definiu *workflow* como 'a automação de um processo de negócio, total ou parcial, na qual documentos, informações ou tarefas são transferidas entre participantes de acordo com um conjunto de regras'.

A definição da WfMC se enquadra bem no contexto dos chamados *Business Workflows*, que traduziremos aqui por Workflows de Negócios. Estes workflows são utilizados no gerenciamento de empresas em suas diversas áreas (estoque, prestação de contas...) para implementar processos de negócio. Nos workflows de negócios os elementos principais são as atividades que fazem parte do processo e a forma de orquestrá-las de forma a atingir um objetivo específico, como, por exemplo, criar uma conta para um cliente.

Além de serem extensamente usados na gestão de empresas os workflows foram adotados também para a modelagem e execução de processos científicos. Segundo [Altintas et al., 2006], um workflow científico é uma combinação de dados e processos em uma sequência estruturada de passos, cujo objetivo é implementar uma solução para um problema científico. Nos workflows científicos sobressaem-se os dados como elementos principais, e por isso costumam ser modelados como *data-flows*, em contraposição aos workflows de negócios que são modelados utilizando principalmente fluxos de controle (control-flow) [Ludascher et al., 2006].

A implementação de um workflow científico pode variar entre manual e totalmente automatizada. Os pesquisadores iniciaram a automatização de seus

workflows através da construção de programas em linguagens de script, como o Perl, muito utilizada em workflows de Bioinformática.

O uso de scripts trouxe várias vantagens já conhecidas da automação de processos, como agilizar a obtenção de resultados, facilitar o processo de repetição do experimento e torná-lo menos sujeito a erros. Porém, existem várias desvantagens no uso de scripts. A mais imediata é a necessidade de aprender uma linguagem de script. Um pesquisador poderia dedicar-se mais a sua atividade fim de pesquisa se não tivesse que aprender a programar. Outro problema é a dificuldade de compreensão do experimento por outro cientista, ou mesmo pelo próprio programador, se este não tomou o cuidado de realizar uma documentação esmerada.

Na seção seguinte apresentaremos os sistemas de gerência de workflow científico, que são utilizados para construir e executar esse tipo de workflow, e possuem diversas vantagens para uso, dentre elas a captura de proveniência de dados.



### 2.1.2. Sistemas de Gerência de Workflow Científico

Os Sistemas de Gerência de Workflow Científico (SGWC) surgiram como uma alternativa aos scripts *ad-hoc*, que perderam um pouco da popularidade para dar lugar a workflows representados nestes sistemas [Freire et al., 2008]. Os SGWC permitem a construção e execução de sequências de tarefas, e em geral disponibilizam uma interface gráfica que abstrai toda ou grande parte da programação que está por trás do fluxo, ajudando o cientista a construir um workflow sem precisar entender de programação. A Figura 1 mostra a interface principal do Taverna [Hull et al., 2006; Oinn et al., 2006], um sistema de workflow científico utilizado principalmente em experimentos de Bioinformática. O workflow exibido foi obtido no site de compartilhamento myExperiment [Goble et al., 2010].

Os SGWC foram desenvolvidos em geral com o enfoque em alguma área de experimentos científicos. O Vistrails [Callahan et al, 2006] oferece uma infra-estrutura direcionada a exploração de dados de imagens e visualização. Já o Taverna possui foco em workflows de Bioinformática, disponibilizando acesso a inúmeros web-services desta área. Neste trabalho utilizamos a ferramenta BioSide [BioSide, 2011] para implementar dois workflows de Bioinformática.

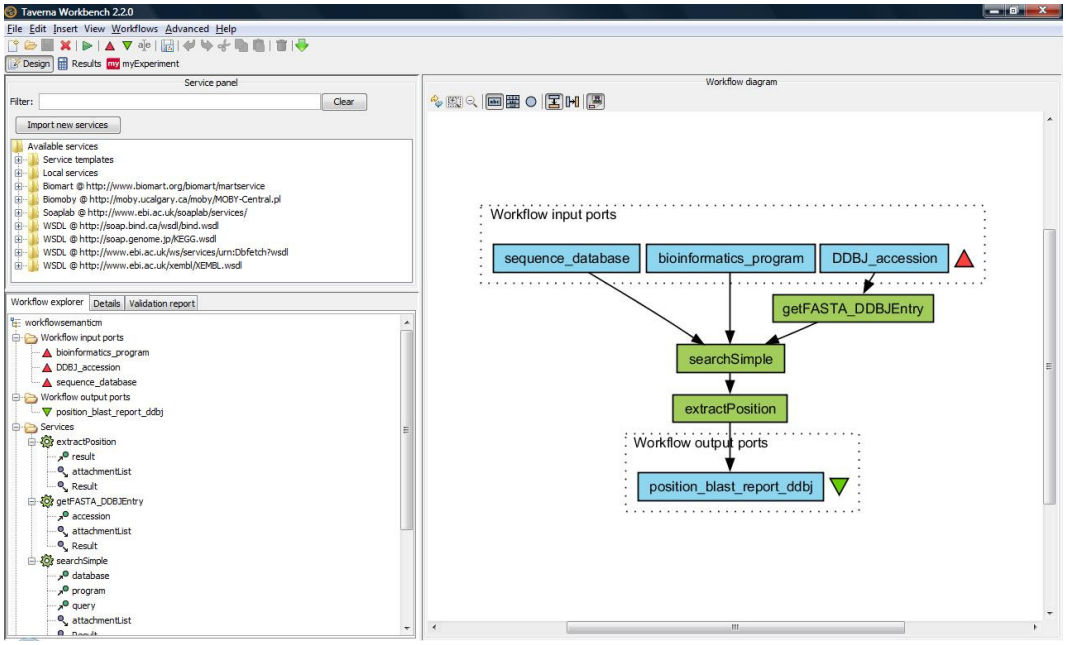


Figura 1 – Interface Principal do Taverna

### 2.1.2.1. Características Gerais dos SGWC

Existem diversos trabalhos que tratam das características dos SGWC existentes. O tutorial [Mattoso e Cruz, 2008] apresenta um panorama sobre os sistemas de workflow científico, suas diferenças para os sistemas comerciais, funcionalidades desejáveis e funcionalidades encontradas nos sistemas, e ainda uma descrição breve dos SGWC mais populares. Em [Curcin e Ghanem, 2008], [McPhillips et al., 2009], [Gil et al., 2007], [Freire et al., 2008], [Deelman et al., 2009], são apresentadas diversas características dos SGWC, assim como comparações entre alguns sistemas.

Aqui será descrito brevemente o funcionamento de um SGWC utilizando a classificação proposta em [Mattoso et al., 2010] para o ciclo de vida de um experimento científico. Nesta classificação, as principais fases de um experimento científico são a *composição*, a *execução* e a *análise*.

Um SGWC provê suporte a essas fases de construção de um experimento de maneira limitada em diversos aspectos, limitações estas apresentadas neste mesmo artigo. As principais limitações levantadas são a falta de suporte a definição de workflows abstratos (seção 2.2.2.8), e a restrição da composição de um experimento científico por um único workflow. A seguir nos limitaremos a apresentar as fases de composição, execução e análise para workflows concretos e não relacionados, uma vez que este trabalho não teve por objetivo tratar as limitações citadas.

#### 2.1.2.1.1. Composição

A composição de um workflow requer o encadeamento de tarefas, a parametrização destas, a definição dos dados de entrada, a inserção de fluxos de controle se necessário. A maioria dos SGWC dispõe de uma interface gráfica para construir o workflow.

Quanto à notação gráfica utilizada para construir o workflow, não há um padrão de descrição adotado pelos sistemas como acontece com os sistemas de modelagem de processos de negócio, que adotaram a notação BPMN [OMG, 2011] como padrão. Cada SGWC utiliza uma simbologia e nomenclatura próprias para representar os passos do workflow, as portas de entrada e saída, as ligações entre os passos etc.

A linguagem utilizada na representação do workflow também varia. O Taverna possui um formato específico chamado t2flow, o Vistrails e o BioSide representam em arquivos XML de schema próprio, o Kepler [Ludascher et al, 2006] usa a linguagem MoML [Lee e Neuendorffer, 2000], o REDUX grava a definição em banco de dados relacional.

Os sistemas disponibilizam uma lista de atividades para a construção do workflow. Na fase de composição o usuário pode escolher as atividades que participarão do workflow, parametrizá-las e concatená-las através de ligações que representam fluxos de dados ou fluxos de controle.

São utilizadas várias nomenclaturas para o que designamos aqui por *atividade*. No Vistrails as atividades são módulos Python, e por isso são chamados apenas *módulos*. O Kepler utiliza o termo *actor*, o Taverna o termo *processor*, enquanto o BioSide se refere a *program*. Iremos usar o termo *atividade* para uniformizar o discurso ao longo do texto.

Ao adicionar uma atividade a um workflow é gerada uma instância desta atividade, com uma parametrização e conexões com outras atividades específicas. Definimos esta instância como *passo de workflow*, ou apenas *passo.* Essa diferenciação de nomenclatura em geral não é utilizada, ou seja, os passos do workflow são comumente chamados pelos sistemas de atividades (ou módulos, ou atores, dependendo do sistema). Consideramos importante essa diferenciação, para ressaltar durante o texto o que é referente à instância da atividade, que participa da definição de um workflow específico, e o que é referente a própria atividade. Na Figura 2 observamos a lista de atividades do

Vistrails e um workflow simples que possui dois passos. O passo *PythonCalc* é uma instanciação da atividade de mesmo nome, assim como o passo *StandardOutput*.

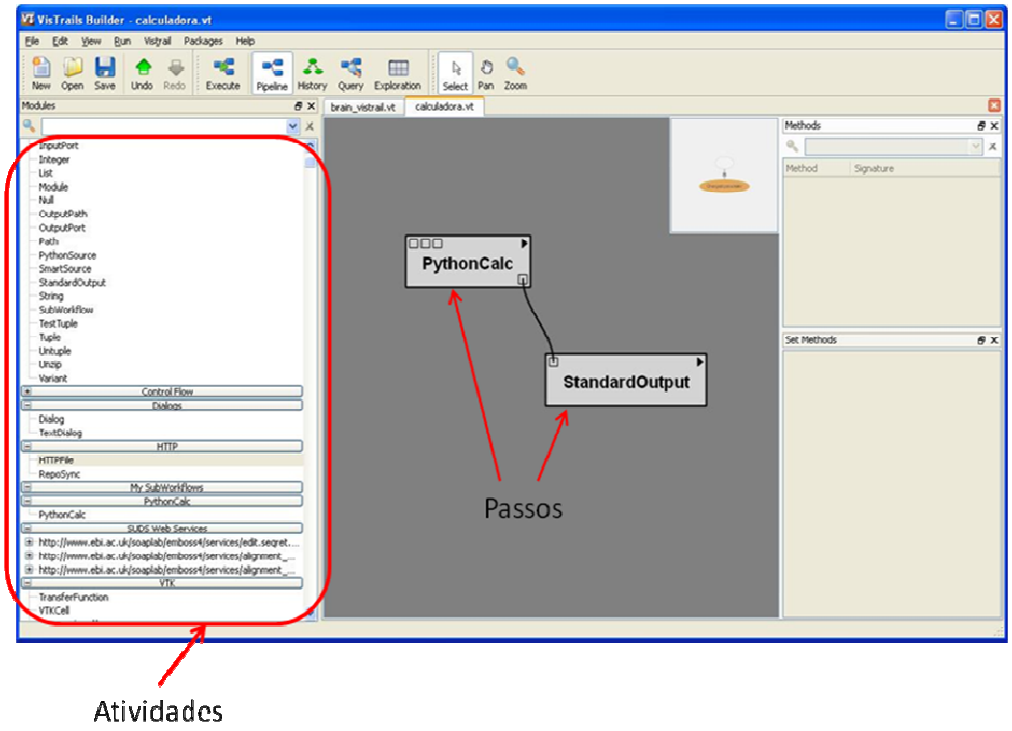


Figura 2 – Atividades e Passos

As atividades podem ou não acessar recursos externos ao sistema, como web services ou aplicativos de linha de comando.

Os sistemas disponibilizam em sua distribuição atividades que funcionam como uma biblioteca de funções locais. Estas atividades são utilizadas por exemplo para tarefas como abertura e manipulação de arquivos, manipulação de strings, interação com o usuário, funções matemáticas, funções de manipulação em banco de dados, dentre muitas outras funcionalidades. Uma característica importante é que em geral essas atividades não acessam nenhuma aplicação externa ao SGWC. Como exemplo podemos citar o módulo de concatenação de strings do Vistrails (ConcatenateStrings).

Já as atividades que acessam recursos externos funcionam como *wrappers*, ou seja, códigos ou descrições estruturadas utilizadas para encapsular a chamada ao programa.

#### 2.1.2.1.2. Execução

Após a escolha e parametrização das atividades que definem os passos do workflow e sua sequência o usuário já pode executá-lo. Durante a execução são gerados logs e dados de proveniência, como tempo de execução dos processos, códigos ou mensagens de erro e dados gerados. Essas informações podem ser persistidas de forma a permitir que o usuário faça consultas posteriormente.

Os SGWC são classificados como locais ou distribuídos, de acordo com o mecanismo de distribuição de tarefas que implementam [Mattoso et al., 2010]. Sistemas locais como Taverna e Vistrails executam os passos do workflow no desktop do cientista, controlando toda a execução de forma centralizada. Na segunda classificação temos sistemas como Pegasus [Deelman et al., 2007] e Triana [Deelman et al., 2007], que permitem a distribuição de atividades e dados em ambientes distribuídos.

#### 2.1.2.1.3. Análise

Na fase de análise de resultados o cientista precisa *consultar* como foi a execução do workflow, quais dados foram gerados, se houve erro na execução de algum processo e outras informações relevantes para que o experimento executado possa ser avaliado.

Nesta fase também sobressai a importância da *reprodutibilidade*, pois em um experimento científico os resultados alcançados podem ser valiosos, mas perdem o valor se não puderem ser reproduzidos. Como provar que um resultado foi obtido através de parâmetros corretos, assim como protocolos e programas adequados? Para validar um resultado é necessário avaliar todo o processo e os dados que o geraram [Freire et al., 2008].

Para possibilitar a avaliação de resultados e a reprodutibilidade o SGWC deve capturar dados de *proveniência*. A captura automática de dados de proveniência é uma das grandes vantagens do uso de sistemas de gerência de workflow.

### 2.1.3. Dados de Proveniência

A palavra proveniência em geral se refere à origem de algo, seja fisicamente tangível ou não. A proveniência de um dado pode ser definida como

o processo que originou o dado, em que se entende por processo as derivações, os bancos de dados, os parâmetros, os artefatos que participaram na geração do dado [Shoshani e Rotem, 2009].

No contexto de banco de dados, a proveniência pode ser caracterizada como a origem de um dado e o processo pelo qual este chegou ao banco de dados [Buneman et al., 2001]. No contexto dos experimentos científicos, existem vários registros que remetem à proveniência de um resultado ou de um processo. Esses registros respondem a perguntas como:

- Qual foi o resultado deste experimento?
- O processo executado está dentro das normas estabelecidas?
- Quais foram os dados utilizados?
- Consigo reproduzir este experimento com estes mesmos dados?

Estas e outras perguntas devem ser respondidas para que se possa validar um experimento, ou seja, verificar se o processo e o resultado obtido são confiáveis [Davidson e Freire, 2008]. Além da confiabilidade, a proveniência auxilia na compreensão do processo científico, no compartilhamento dos resultados e no reuso de componentes prontos.



#### 2.1.3.1. Classificações de Proveniência

Alguns trabalhos apresentam classificações de proveniência para o domínio particular dos workflows científicos. Uma das primeiras classificações ocorreu em [Greenwood et al., 2003], que considera duas formas predominantes de proveniência: derivação e anotação. A derivação consiste em registrar o processo pelo qual os dados passam para gerar um resultado, o que pode incluir, por exemplo, parâmetros de entrada, bancos de dados utilizados, queries em um banco de dados, definição de um workflow. As anotações consistem em informações descritivas sobre objetos, como data de criação, data de atualização e formato, assim como anotações mais complexas que descrevem, por exemplo, o significado do objeto no domínio ao qual ele pertence.

Já em [Zhao et al., 2006] e [Clifford et al., 2008] a proveniência é classificada em duas grandes vertentes, a prospectiva e a retrospectiva. A proveniência prospectiva corresponde à definição do procedimento que deve ser seguido para gerar um determinado dado. Já a retrospectiva corresponde a todas as informações de execução do processo de geração do dado, como

dados de entrada e saída e outras informações referentes à execução do processo propriamente dita.

Em [Freire et al., 2008] a classificação da proveniência em prospectiva e retrospectiva continua sendo utilizada, mas são destacadas mais duas definições importantes de proveniência: a causalidade e as informações definidas pelo usuário. A causalidade se refere às ligações entre processos e dados ou entre dados, que podem ser inferidas a partir dos registros de proveniência prospectiva e retrospectiva. As informações definidas pelo usuário são classificadas como informações que não são capturadas automaticamente, mas registram importantes comentários e decisões. São as chamadas anotações, termo também adotado em [Greenwood et al., 2003], que podem ser associadas tanto a registros de proveniência prospectiva como retrospectiva.

Uma taxonomia específica para proveniência em SGWC foi elaborada em [Cruz et al., 2009], com base em [Simmhan et al., 2005], que propôs uma classificação de proveniência genérica para problemas computacionais.

Tomando por base todas as classificações encontradas, pode-se definir *proveniência prospectiva* como a especificação de um workflow, que consiste no conjunto de passos do mesmo, parâmetros e dados de entrada utilizados e conexões entre os passos. Já a *proveniência retrospectiva* se refere aos dados de execução de um workflow, que consiste nos dados produzidos e informações sobre a execução de cada passo (tempo de execução, mensagens de erro, local de execução). Situando a captura destes dados no contexto do ciclo de vida de um experimento científico, a proveniência prospectiva é capturada na fase de composição e a retrospectiva na fase de execução [Cruz et al., 2009].



### 2.1.3.2. Open Provenance Model

O OPM (Open Provenance Model) é um modelo de representação de *proveniência retrospectiva* cuja criação foi motivada pelos *Provenance Challenges* [ProvChallenges, 2011], desafios propostos para que os diversos sistemas de proveniência existentes pudessem se auto-avaliar e expor suas características, dificuldades e limitações ao tentar responder algumas consultas de proveniência. Houve um consenso após o 2º desafio de que era necessária uma linguagem comum para que os sistemas pudessem compreender-se mutuamente. Neste cenário surgiu o OPM em sua primeira versão. Em 2010 foi

publicada a versão 1.1 [Moreau et al., 2010] do modelo, que foi a versão considerada nesta dissertação.

Alguns objetivos do OPM são: Permitir que informações de proveniência sejam trocadas entre os diversos sistemas, permitir a representação da proveniência de uma maneira independente de tecnologia, e permitir que os desenvolvedores de sistemas possam construir ferramentas que estejam baseadas de alguma forma no modelo.

Dado um conjunto de elementos que possuem uma relação de dependência, o OPM propõe a construção de um grafo dirigido que representa as dependências entre esses elementos. Estes podem ser de acordo com o modelo *artefatos*, *processos* ou *agentes*, que constituem os nós do grafo de proveniência. As ligações entre os nós são definidas como dependências, que podem ser dos seguintes tipos: used (R), wasGeneratedBy (R), wasControlledBy (R), wasTriggeredBy, wasDerivedFrom.

A origem da seta indica o efeito e o destino a causa. A Figura 3 exibe as ligações que podem ser feitas entre artefatos para formar o modelo. Pode-se incluir um texto entre parênteses, junto às dependências used, wasGeneratedBy e wasControlledBy. Este texto, representado por 'R', indica a 'Role' pela qual o artefato foi usado ou gerado, ou ainda a 'Role' de um agente influenciador de um processo. A 'Role' é uma forma de representar diferentes funções de artefatos e agentes em relação aos processos.

Utilizando este modelo podemos representar o histórico de um dado: Quais processos foram responsáveis pela sua criação, quais processos o utilizaram, de quais artefatos foi derivado, quais 'agentes' influenciaram no processo. Outras informações podem ser inferidas pelas regras de inferência disponibilizadas pelo modelo.

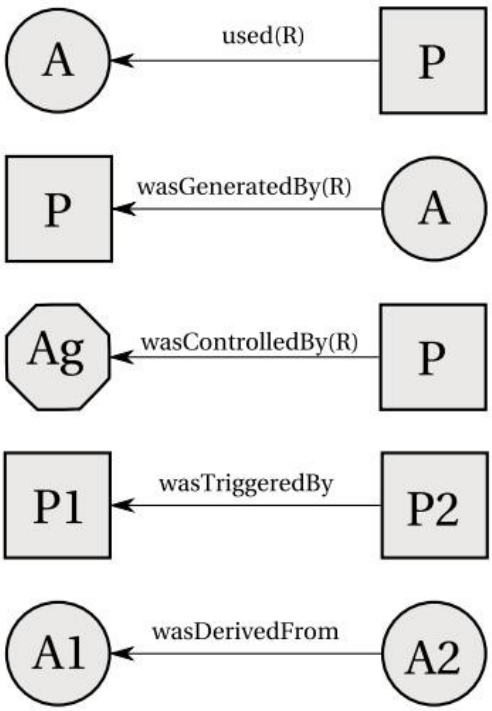


Figura 3 – Elementos e dependências que podem participar de um grafo OPM



### 2.1.3.3. Sistemas e modelos de proveniência

Nesta seção apresentaremos brevemente as características gerais dos modelos de proveniência pesquisados neste trabalho. Além do BioSide, utilizado na implementação do projeto de proveniência, foram escolhidos alguns dos sistemas de workflow científico mais usados atualmente (Vistrails, Taverna, Kepler). Descrevemos também o modelo do REDUX, que representa todos os dados de proveniência em modelo relacional, e o modelo do e-Bioflow, que utiliza OPM em modelagem relacional, tal como o modelo proposto neste trabalho.

#### 2.1.3.3.1. Vistrails

O Vistrails é um SGWC construído com foco em experimentos de simulação, exploração de dados e visualização, nas quais o usuário constrói workflows de diversas formas para ao final compará-las.

Embora tenha grande destaque a união entre as funcionalidades de exploração de dados e visualização, o sistema pode ser utilizado para a criação de outros tipos de workflows científicos, inclusive de bioinformática. Para tal

devem ser adicionados os módulos necessários que não vêm embutidos na ferramenta e devem ser programados em Python no formato definido pelo sistema.

Enquanto alguns sistemas como Taverna e Kepler tiveram que ser estendidos para prover funcionalidades de proveniência, o Vistrails as incorporou desde o início do seu desenvolvimento [Freire et al., 2008]. O modelo de proveniência do sistema é definido em [Callahan et al., 2006] como *action-oriented*. Durante a construção do workflow, cada alteração realizada pelo usuário é registrada, como em um log de transações de banco de dados [Ellkvist et al., 2008], sendo instanciado um novo nó no histórico de alterações (Figura 4). Dessa forma o sistema grava apenas a diferença entre uma versão e outra e precisa apenas aplicar as diferenças para caminhar entre os nós da árvore, que representam cada um uma versão do workflow. Isso promove para o usuário a visualização de todas as diferentes versões obtidas, explorando variações no design de seu experimento, com um baixo custo de armazenamento.



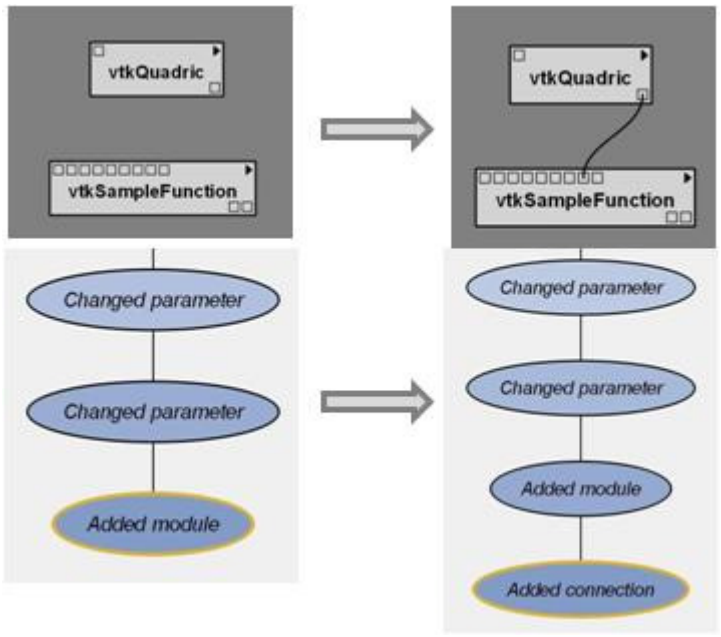


Figura 4 – Geração de uma versão de workflow para cada alteração

O Vistrails captura as definições e execuções dos workflows armazenando-as em arquivos XML ou em banco de dados relacional. O banco de dados é usado apenas como estrutura de persistência, não tendo sido projetado para oferecer ao usuário consultas SQL diretas ao banco. Consultas sobre a definição de workflows gerados em um arquivo *.vt*, como é chamado o arquivo de especificação do workflow, podem ser feitas através da funcionalidade de

consulta por exemplo [Scheidegger et al., 2008], uma forma de consulta visual na qual o usuário descreve a consulta da mesma forma que constrói o workflow. Um exemplo seria consultar quais workflows (nós da árvore de histórico) contém um módulo específico. Para tal basta adicionar o módulo na interface de consulta e executar. O sistema exibirá os nós que possuem aquele módulo.

As consultas sobre a definição ou o histórico de alterações podem ser realizadas também em uma linguagem textual específica do Vistrails, criada para facilitar consultas de proveniência [Vistrails User Guide, 2011]. Entretanto este método provê consultas bastante limitadas. Consultas em mais de um .vt não são possíveis utilizando nem a linguagem visual nem a linguagem textual de consulta [Koop, 2011c]. Já consultas sobre dados de execução são mais complicadas. No FAQ [Vistrails FAQ, 2011] temos um exemplo de consulta ao log de execução de um .vt (Figura 5), que usa python e funções de acesso ao log para retornar alguns dados das execuções realizadas, como tempo de execução e módulos executados.



```
import core.log.log
import db.services.io
```

```
def run(fname):
 # open the .vt bundle specified by the filename "fname"
 bundle = db.services.io.open_vistrail_bundle_from_zip_xml(fname)[0]
 # get the log filename
 log_fname = bundle.vistrail.db_log_filename
 if log_fname is not None:
     # open the log
     log = db.services.io.open_log_from_xml(log_fname, True)
     # convert the log from a db object
     core.log.log.Log.convert(log)
     for workflow_exec in log.workflow_execs:
         print 'workflow version:', workflow_exec.parent_version
         print 'time started:', workflow_exec.ts_start
         print 'time ended:', workflow_exec.ts_end
         print 'modules executed:', [i.module_id
                                     for i in workflow_exec.item_execs]
if __name__ == '__main__':
   run("some_vistrail.vt")
```

Figura 5 – Consulta ao log do Vistrails para listar módulos executados

### 2.1.3.3.2. Kepler

O Kepler [Ludascher et al, 2006] é um SGWC construído como uma extensão do framework Ptolemy [Ptolemy, 2011], que promove a construção de data-flows baseados na conexão de componentes de software chamados *atores*. No Ptolemy a semântica da comunicação entre os atores é definida por um

componente chamado *diretor*, que descreve sob qual modelo de computação o workflow será executado.

Inicialmente o Kepler armazenava a definição dos workflows em uma linguagem de modelagem baseada em XML chamada MoML, e não disponibilizava nenhum tipo de registro de proveniência. Na versão 2.0 o sistema incluiu várias funcionalidades de proveniência, principalmente o registro de execuções e definições em banco de dados relacional.

A gravação dos dados de proveniência em banco está habilitada por padrão na distribuição da ferramenta, sendo que o usuário pode optar pelo desligamento dessa funcionalidade a qualquer momento via interface (botão de habilitar/desabilitar proveniência). Existe um arquivo de configuração disponível para edição do usuário que informa o banco de dados a ser utilizado (atualmente por padrão é o HSQLDB [HSQLDB, 2011], mas também pode ser utilizado o mySQL [MySQL, 2011]), usuário e senha de acesso, e algumas opções de detalhamento sobre a captura da proveniência. Caso o usuário queira efetuar uma gravação específica para um determinado workflow pode-se utilizar um componente chamado Provenance Recorder, que deve ser adicionado ao workflow e configurado. Cabe observar que esta configuração ficará válida apenas para este workflow.

O esquema de dados (Figura 6), descrito em [Kepler Prov, 2010], possui a representação de alguns componentes do workflow, como atores, portas e parâmetros, as execuções do mesmo, e também uma entidade criada para registrar alterações no workflow. Para a definição do fluxo em si continua sendo utilizada a linguagem MoML, e por isso o arquivo MoML do workflow deve ser gravado para garantir a reprodutibilidade do mesmo.

Os dados armazenados podem ser consultados usando SQL ou uma API *java* que disponibiliza algumas consultas [Kepler Prov, 2010].

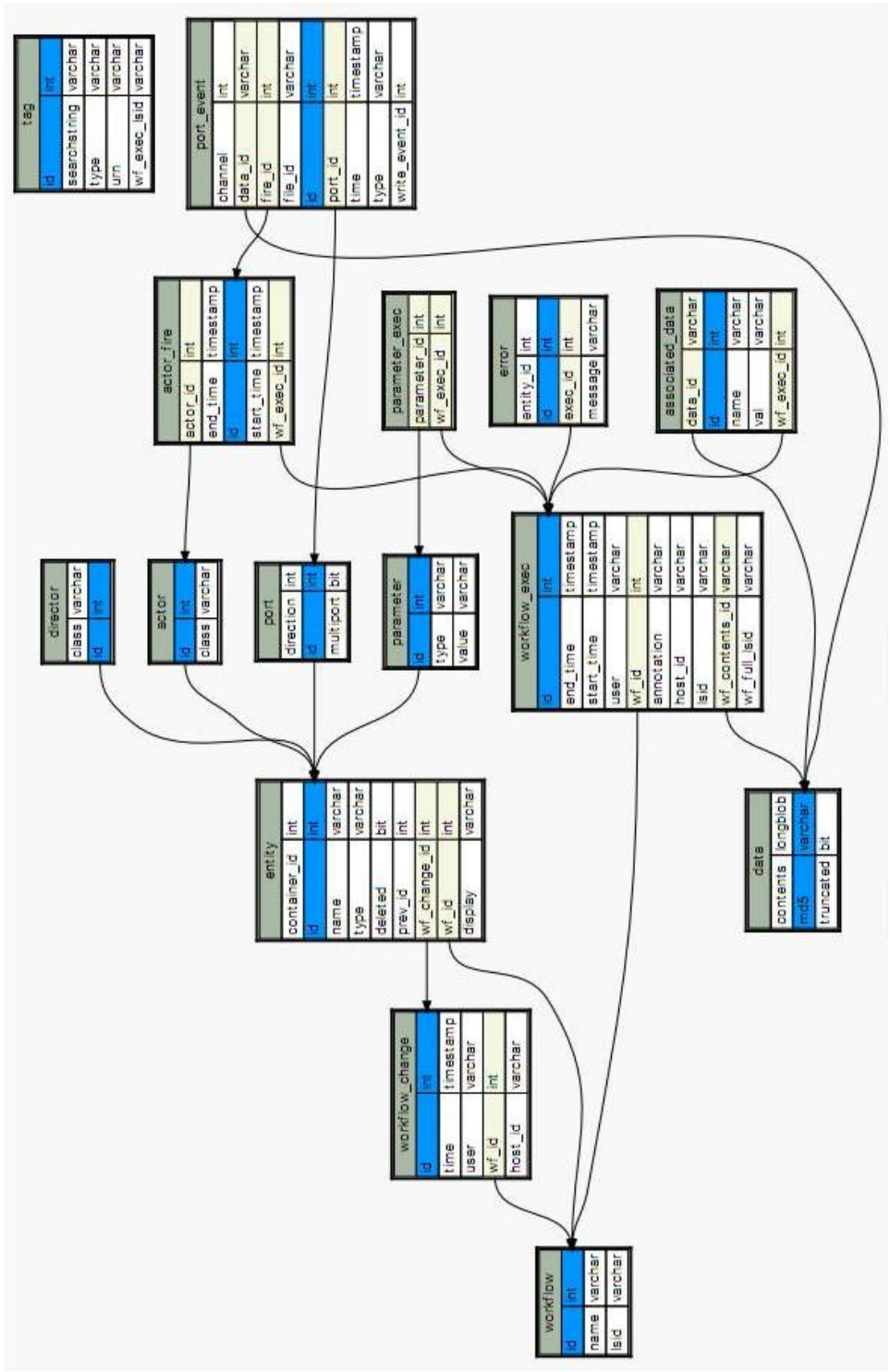


Figura 6 – Esquema de proveniência do Kepler

#### 2.1.3.3.3. Taverna

O Taverna é uma ferramenta desenvolvida pelo projeto myGrid [myGrid, 2011] que permite a construção e execução de workflows através da composição de web services. Qualquer web service que possua uma descrição WSDL pode ser adicionado à lista de serviços disponíveis do Taverna. Outros tipos de serviços também podem ser adicionados, como serviços BioMoby e Soaplab. Embora esteja sendo bastante utilizado no domínio da Bioinformática devido a enorme disponibilidade de serviços dessa área, seu uso não está restrito a nenhum domínio.

Os workflows são representados em uma linguagem baseada em XML, que evoluiu do formato Scufl (Simple conceptual unified flow language) para o formato t2flow nas versões a partir da 2.0.

O esquema de dados (Figura 7), disponível em [Taverna Schema, 2011], representa alguns dados de proveniência prospectiva e retrospectiva, sendo que tal como no Kepler, a definição completa do workflow continua sendo gravada no arquivo de definição, neste caso no formato t2flow. O arquivo é gravado em hexadecimal no campo dataflow da tabela Workflow.

O Taverna registra dados de proveniência em banco de dados relacional, porém o banco não foi desenvolvido para que o usuário final consulte a proveniência [Soiland-Reyes, 2010]. Uma linguagem de consulta está em desenvolvimento [Taverna Query, 2011] para implementação na versão 2.3. Além de fazer consultas nessa linguagem o usuário poderá exportar dados para o formato RDF e OPM [Soiland-Reyes, 2010]. A persistência em banco da proveniência é opcional, o usuário pode optar por manter os dados apenas em cache. Neste caso os dados serão apagados ao sair da ferramenta.

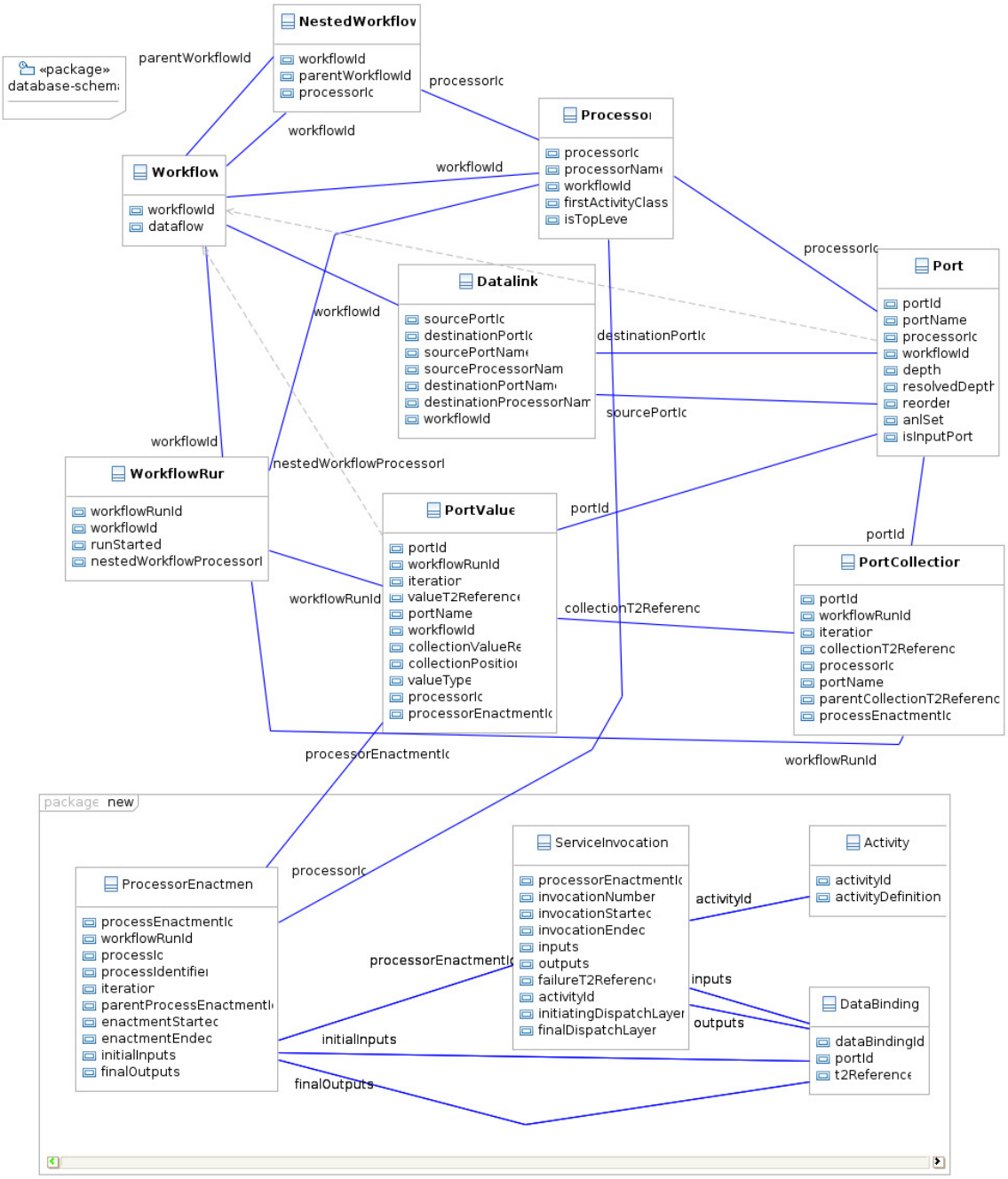


Figura 7 – Esquema de dados de proveniência do Taverna

## 2.1.3.3.4. BioSide

O BioSide [BioSide, 2011] é uma ferramenta que foi desenvolvida com o foco na construção de workflows de Bioinformática. São disponibilizadas por padrão no sistema atividades de acesso a programas de linha de comando, a maioria relacionada a processos de Filogenia.

O modelo de computação adotado é um data-flow simples, no qual quando todos os dados de entrada de um passo do workflow estão disponíveis o mesmo é executado.

No BioSide, para cada execução de workflow é gerado um diretório contendo a definição executada em um arquivo XML e os arquivos consumidos e produzidos. Todos os arquivos são copiados para este diretório. Para cada execução é gerado um log também em formato XML, e logs específicos dos passos executados, que são gerados em arquivo texto. O BioSide exibe visualmente as execuções realizadas, mas não é possível realizar nenhum tipo de consulta sobre os dados.

#### 2.1.3.3.5. REDUX

O REDUX [Barga e Digiampietri, 2008] é um sistema de proveniência desenvolvido para a ferramenta de execução de workflows Windows Workflow Engine [WinWorkflow, 2011]. A principal característica do seu modelo de proveniência é a divisão em níveis de abstração, que abrange desde o nível conceitual da definição das atividades até o nível de execução.

No primeiro nível, chamado L0, é representada a definição abstrata do workflow, composta por atividades abstratas como 'tarefa de alinhamento de sequências'. Esse nível permite fazer consultas do tipo 'Quantas tarefas de alinhamento são utilizadas nos meus workflows?'. Já no segundo nível cada tarefa abstrata é mapeada para uma atividade específica, por exemplo, 'web-service BLAST NCBI'. O terceiro e quarto níveis registram a execução do experimento, sendo que no terceiro tem-se por exemplo dados de entrada, parâmetros e caminhos escolhidos e, no quarto, detalhes mais operacionais como duração das tarefas e códigos de saída.

O modelo relacional é utilizado, sendo que sistema permite dois tipos de modelagem, uma chamada 'single provenance relation', que para cada nível constrói apenas uma tabela com todos os dados, e a outra 'multiple provenance relations', que constrói várias relações em cada nível. A Figura 8 exibe os esquemas dos níveis L0 e L2 na opção de múltiplas tabelas.

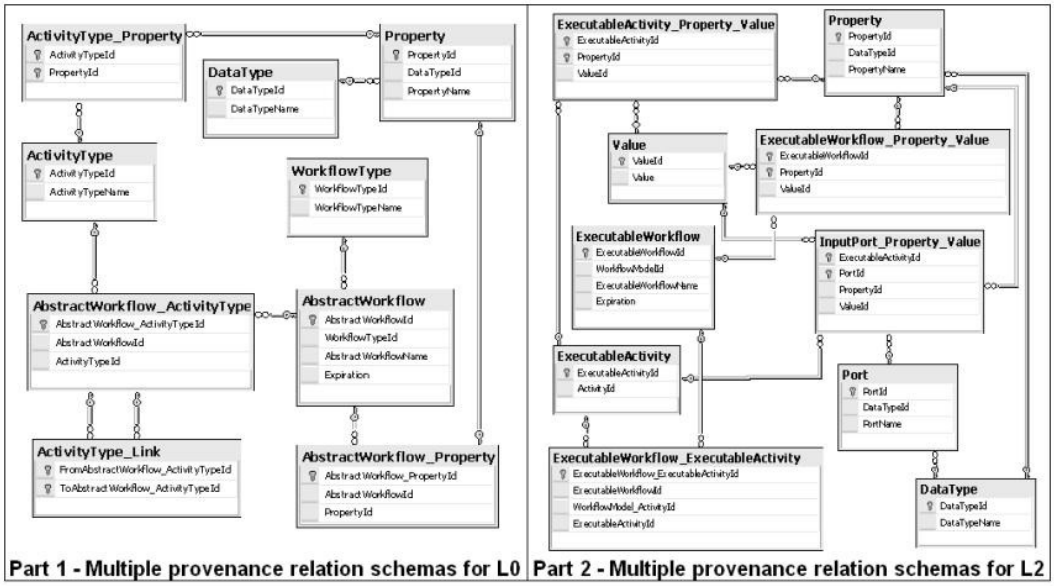


Figura 8 – Níveis L0 e L2 dos esquemas do REDUX

### 2.1.3.3.6. e-BioFlow

O desenvolvimento do e-BioFlow [Wassink et al., 2010] foi motivado pela necessidade de abordar como alguns problemas dos sistemas de workflow existentes poderiam ser tratados [Wassink, 2010], principalmente a interação com o usuário e algumas questões sobre proveniência. O processo de instalação do sistema e configuração do ambiente para a captura de dados de proveniência não é trivial, visto que não há documentação suficiente.

Optamos por apenas apresentar o esquema utilizado, que possui algumas semelhanças com o esquema descrito neste trabalho.

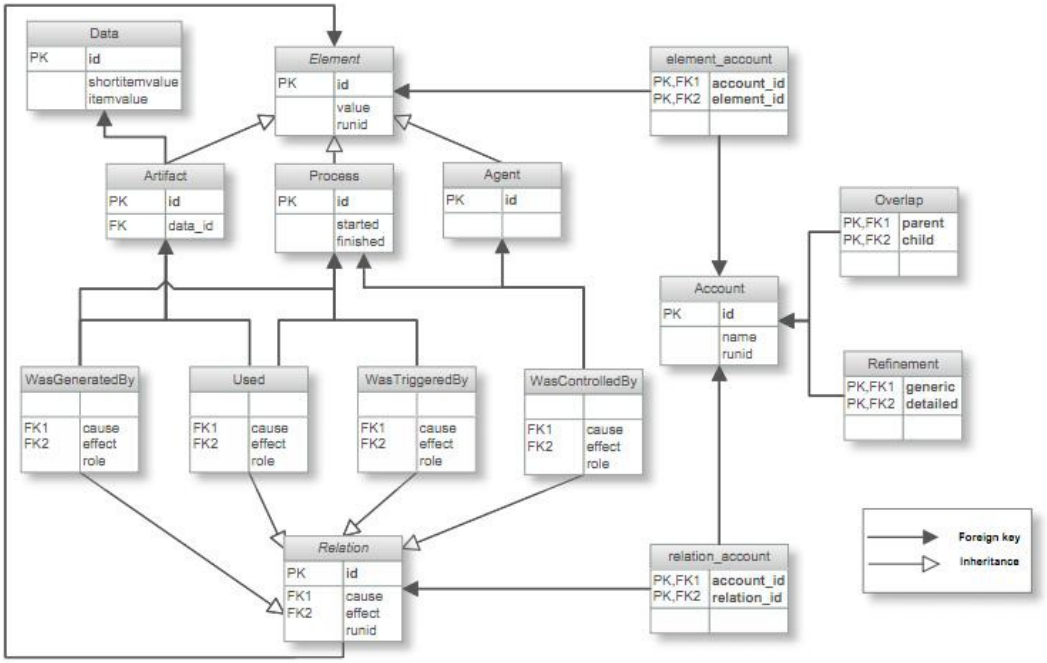


Figura 9 – Esquema relacional do e-BioFlow, obtido em [Ooms, 2009]

O esquema (Figura 9) proposto no e-BioFlow representa as entidades do OPM (na versão 1.01) como entidades no modelo relacional. Foram criadas duas entidades genéricas *Element* e *Relation*, para facilitar consultas em nível genérico. A tabela *Data* armazena os dados gerados e consumidos pelas tarefas, e os artefatos apenas referenciam os dados armazenados em *Data*.

## 2.2. Motivação

Nesta seção discutiremos algumas características importantes de workflows de Bioinformática, e levantaremos alguns desafios de proveniência que encontramos na literatura e no estudo de alguns SGWC. Daremos destaque ao desafio da reprodutibilidade, para a qual criamos uma classificação em dois níveis. Listaremos enfim alguns objetivos do presente trabalho.

### 2.2.1. Workflows em Bioinformática

Apresentaremos aqui em alto nível os dois workflows utilizados nos estudos de caso.

O workflow MHOLline (www.mholline.lncc.br) [Capriles et al., 2010] foi desenvolvido pelo Grupo de Modelagem Molecular de Sistemas Biológicos (GMMSB) do LNCC/MCT[1] e pelo o Laboratório de Física Biológica (FISBIO) do IBCCF-UFRJ/ME[2], e atualmente é disponibilizado como serviço na web por esta última instituição.

O MHOLline é um workflow científico utilizado para a tarefa de modelagem comparativa da estrutura terciária de proteínas. O workflow (Figura 10) recebe como entrada um arquivo no formato FASTA contendo uma ou mais sequências de aminoácidos. Cada sequência de entrada será alinhada com todas as proteínas do banco de dados PDB (Protein Data Bank) [Berman et al., 2000] através do programa BLAST. O próximo passo é a execução do programa BATS, que atribui de forma heurística uma pontuação a cada alinhamento obtido anteriormente, classificando também qual a sequência do PDB que gerou o melhor alinhamento. Essa sequência é chamada de *champion sequence*, e será a única utilizada para gerar o modelo terciário da sequência de entrada.

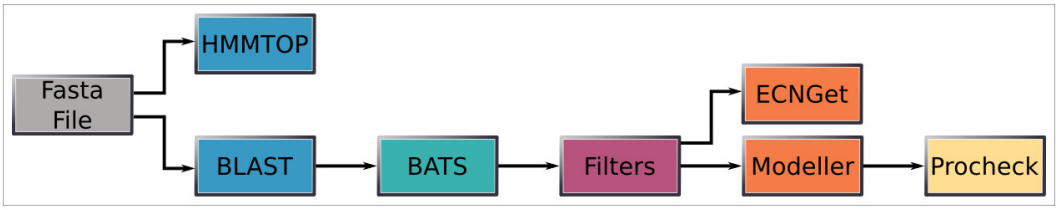


Figura 10 – Workflow MHOLline

O outro workflow utilizado como estudo de caso é um workflow (Figura 11) que tem por objetivo construir árvores filogenéticas a partir de sequências de proteínas. Este workflow foi obtido da seção de exemplos do site do sistema BioSide.

[1] Laboratório Nacional de Computação Científica / Ministério da Ciência e Tecnologia

[2] Instituto de Biofísica Carlos Chagas Filho – Universidade Federal do Rio de Janeiro / Ministério da Educação

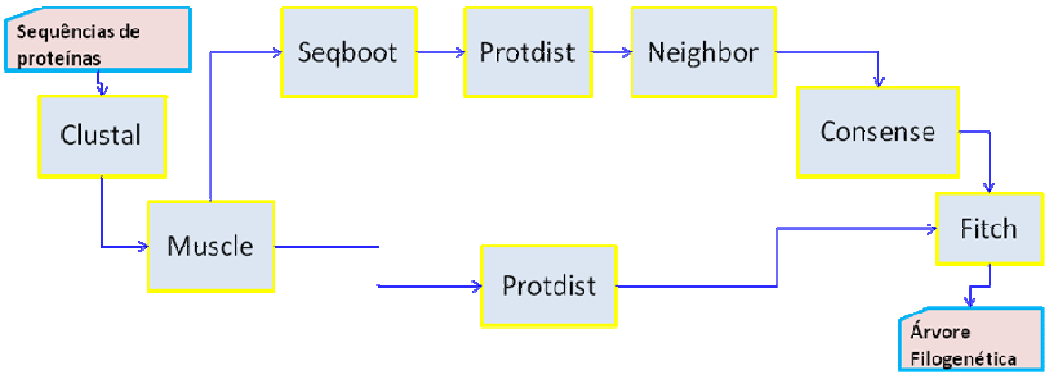


Figura 11 – Workflow de geração de árvores filogenéticas

Os workflows de Bioinformática podem ser classificados como data-flows uma vez que os dados são os elementos principais do processo, sobre os quais são aplicados filtros e transformações, para chegar a outros dados de interesse do cientista [Taylor et al., 2007].

Uma caracterização de um workflow de Bioinformática de forma completa necessitaria do estudo de workflows utilizados em cada área da Biologia, suas características comuns e diferenças, componentes e tipos de dados acessados, freqüência do uso de estruturas de controle e outros aspectos. Porém podemos apontar o que, em geral, não aparece em workflows de Bioinformática, que são o acesso direto a funções matemáticas ou o uso de interação com sensores, funcionalidades que aparecem no Kepler e Triana, por exemplo.

Para atender as necessidades da Biologia, foram desenvolvidos alguns sistemas de workflow que oferecem tarefas de bioinformática. É o caso do Taverna, que disponibiliza acesso a milhares de serviços, sendo uma boa alternativa para a construção de workflows baseados em orquestração de web services.

Porém, os workflows de Bioinformática não se resumem a uma composição de web services. Em [Wassink et al., 2009], há um levantamento de 415 workflows construídos no Taverna e disponíveis no site MyExperiment, indicando a freqüência de uso de ‘Local Services’, ‘web services’, ‘Scripting’ e ‘sub-workflows’. Nesta classificação, ‘Local Services’ são definidos como tarefas executadas localmente pelo próprio sistema de workflow, e ‘Scripting’ são scripts programados pelo usuário e inseridos usando os serviços de execução de scripts, como o beanshell. Os sub-workflows são workflows completos invocados dentro de outro workflow, em geral para fins de reuso ou para tornar mais compreensíveis workflows muito grandes. Ficou constatado no mesmo trabalho que 57% das tarefas eram serviços locais, 22% eram web-services, 14% scripts,

e o restante sub-workflows. Dos serviços locais 53% são tarefas de conversão de dados, que também aparecem como scripts, porém sem especificação da porcentagem.

As tarefas de conversão de dados são muito comuns na Bioinformática devido à pluralidade de formatos de dados existentes. Para interligar dois serviços muitas vezes o cientista precisa inserir uma tarefa que converte a saída do primeiro no formato esperado pelo segundo. Na pesquisa citada, o autor aponta para a notável freqüência dessas tarefas, que somam mais de 30% do total analisado, o que indica que os cientistas gastam muito tempo e esforços com tarefas que não fazem parte da linha principal do experimento, e que em sua maior parte não são web services.

Além desses scripts e tarefas locais de conversão de dados, inúmeros programas de linha de comando são comumente usados na Bioinformática e podem ser necessários na construção do fluxo principal de um workflow. Os dois workflows apresentados anteriormente utilizam apenas programas stand-alone. Existem também aplicações de análise de grandes volumes de dados que devem ser executadas localmente devido ao overhead de tráfego que gerariam se fossem web services. Em [Benson, 2010] foi incluída a aceitação de submissão de aplicações “non-web Server” em atenção à necessidade de se executar localmente determinados tipos de tarefas, que portanto não necessitam ser implementadas como serviços web.

O uso de aplicações de linha de comando na Bioinformática levanta o desafio do registro de proveniência em relação a esses programas. Como descrevê-los, que metadados armazenar em relação a definição e execução de um workflow que os utiliza, enfim, como tratar a sua proveniência de dados.

Além de programas stand-alone, outra característica comumente encontrada nos experimentos de Bioinformática é o uso de bases de dados e arquivos disponibilizados localmente, muitas vezes no desktop do cientista. Alguns desafios listados na próxima seção foram descritos com base neste contexto de experimento, que motivou também a elaboração do projeto de proveniência proposto neste trabalho.

### 2.2.2. Desafios de Proveniência

O armazenamento de informações de proveniência de dados possui diversas questões interessantes de pesquisa. Algumas dessas questões foram exploradas nos Provenance Challenges.

No primeiro desafio o objetivo era compreender as capacidades de cada sistema em relação à captura e consulta de proveniência. Foi proposto um workflow que os sistemas participantes deveriam implementar e realizar as consultas estipuladas.

No segundo desafio o objetivo era compreender as dificuldades de interoperabilidade entre os sistemas. Foram colocadas questões sobre a importação e exportação de dados de proveniência entre sistemas e a capacidade de um sistema compreender um workflow e resultados de outro sistema. O aprendizado deste desafio incentivou a criação do OPM, para que os sistemas pudessem compreender informações geradas por outros sistemas.

No terceiro desafio a proposta foi usar a 1ª versão do OPM para efetuar a interoperabilidade entre sistemas. Este desafio gerou melhorias propostas para o OPM que serviram de base para o aperfeiçoamento do modelo, que hoje está na versão 1.1 [Moreau et al., 2010].

O quarto desafio propõe a aplicação do OPM em um cenário de experimentos mais abrangente, que extrapola o uso exclusivo de sistemas de workflow.

Apresentaremos a seguir alguns dos desafios motivados nos Provenance Challenges e outros encontrados através da pesquisa realizada neste trabalho.

#### 2.2.2.1. Reprodutibilidade

A reprodutibilidade é um dos principais objetivos do armazenamento de dados de proveniência. Um SGWC deve permitir que seja possível reproduzir o mesmo experimento que gerou um resultado interessante em uma pesquisa científica. Os meios de publicação estão exigindo cada vez mais a capacidade de reproduzir o processo científico que se quer publicar [Mattoso e Cruz, 2008].

Um problema ao se tratar deste desafio é que a palavra *reprodutibilidade* pode ter diferentes interpretações dependendo do contexto adotado. Quando se fala *reproduzir um workflow*, pode-se estar referindo a reprodução da definição do processo, sem preocupação de usar os mesmos dados ou mesmas versões

de programas usados em alguma execução prévia. Já para garantir a confiabilidade de uma publicação de bioinformática, relativa a um tipo de análise de sequências biológicas, pode ser necessário que o experimento possa ser reproduzido utilizando a mesma especificação de workflow e os mesmos dados de entrada, e assim gerar os mesmos resultados. Em outros tipos de experimentos, como por exemplo o sistema BioProvider [Noronha, 2006], será importante gravar e poder reproduzir dados como alocação em memória, tráfego na rede, e outras características intrínsecas ao ambiente no qual o experimento foi realizado.

Cabe ressaltar que o fato de um sistema prover algumas funcionalidades de captura de proveniência prospectiva e retrospectiva não significa que será possível reproduzir exatamente a mesma especificação de workflow, ou seja, as mesmas versões de atividades usadas no momento da execução. Isto irá depender do nível de granularidade dos dados registrados pelo sistema ao capturar a proveniência. O problema se torna ainda mais complexo quando as atividades acessam recursos externos ao sistema, como programas de linha de comando ou web services.

Em suma, reproduzir um experimento pode se tornar um desafio dependendo do grau de reprodutibilidade que se quer atingir. Para que fique claro ao longo do texto sobre que tipo de reprodutibilidade estamos falando, optamos por classificar na Seção 2.2.3 dois níveis de reprodutibilidade explicitando algumas informações que podem ser capturadas pelo SGWC referentes a cada nível.



#### 2.2.2.2. Gerência de Dados Consumidos e Produzidos

Ao compor um workflow, o usuário define dados de entrada para os processos, que podemos chamar de *dados consumidos*. Na fase de execução, serão gerados dados intermediários e finais, chamados de *dados produzidos*.

A gravação dos dados consumidos e produzidos pode ser importante para atender o requisito de reprodutibilidade, caso se deseje utilizar os mesmos dados do momento da execução. Note-se que registrar apenas uma referência para o dado como o caminho aonde ele se encontra é insuficiente uma vez que o arquivo pode sofrer modificações entre uma execução e outra. Se o arquivo é um banco de sequências biológicas, por exemplo, o usuário pode atualizar a

versão do banco sem que o sistema tome conhecimento desta atualização, uma vez que só foi gravado o caminho e nome do arquivo.

Esta gravação de dados se torna um desafio na área da Bioinformática, que frequentemente utiliza grandes bases de dados biológicos ou grandes arquivos de entrada, inviabilizando a gravação destes dados de forma freqüente, por exemplo, em cada execução.

No sistema Kepler, os dados consumidos e produzidos são armazenados no banco. O dado original é convertido em hexadecimal e gravado, assim como o md5 do dado, que é utilizado como identificador único do próprio. Para o uso de programas de Bioinformática que gerem ou consumam grandes arquivos de dados, a gravação em banco relacional de tudo é inviável. Não há tratamento no Kepler para esse tipo de problema, e a orientação dada [Jones, 2011a] para os casos em que o usuário deve executar um programa local que produza um arquivo grande é que este seja gerado em um diretório convencional, sem nenhum tipo de controle de proveniência.

Já no Vistrails, em [Koop et al., 2010b] é descrito um framework para fazer gerência de dados de entrada, saída e dados intermediários, que foi incluído na versão 1.6 do sistema. A implementação constitui em um pacote de persistência, que oferece módulos que devem ser adicionados a definição do workflow para que os arquivos de entrada e saída sejam persistidos mesmo após o fechamento do sistema. Os arquivos permanecem assim sob gerenciamento do próprio sistema. O uso deste pacote promove também o versionamento transparente de arquivos, gerando uma nova versão para um arquivo que tenha sido alterado. A implementação da persistência de arquivos usa git [git, 2011] para o armazenamento dos arquivos e sqlite [SQLite, 2011] para armazenar metadados dos mesmos. O framework permite que em uma re-execução apenas sejam gravados novamente dados que sofreram modificação.

No SGWC Taverna, tal como no Kepler, todos os dados gerados e consumidos são gravados em banco relacional. A diferença do Taverna é que os dados são sempre gravados, mesmo que já exista no banco um registro idêntico. Não há tratamento para o caso de grandes arquivos de dados que podem ser usados como entrada ou gerados como saída de processos.

### 2.2.2.3. Reuso de Dados

Uma das grandes vantagens do surgimento dos SGWC foi a possibilidade de compartilhar workflows entre usuários, via sites de compartilhamento como o MyExperiment. No Taverna, por exemplo, o pesquisador pode importar workflows já existentes e interligá-los construindo um novo experimento. Estes workflows estarão conceitualmente unidos em uma nova definição de workflow que poderá evoluir independentemente das definições originais dos workflows importados.

Porém existe outra possibilidade de construção de experimentos nos quais ocorre a participação de mais de um workflow, baseada no reuso de produtos intermediários ou finais. Um workflow genérico W1 pode ser executado várias vezes, gerando várias saídas diferenciadas. Se uma dessas saídas for usada como entrada de um outro workflow W2 em alguma ou várias execuções, estas terão uma ligação com o workflow W1.

Um exemplo prático deste problema pode ser observado em um possível cenário de uso do workflow MHOLline (Figura 12). Foi levantado um requisito específico pela instituição provedora do MHOLline, que consiste em permitir ao usuário a escolha de outros templates além do escolhido pelo programa BATS para gerar o modelo. O BATS escolhe sempre o template de maior pontuação, e no caso de empate possui uma lógica própria para escolher apenas um template. Seria interessante fornecer ao usuário a possibilidade de testar outro template, que teve a mesma pontuação ou mesmo uma pontuação inferior, pois o modelo gerado poderia eventualmente ser melhor do que com a escolha padrão do BATS, visto que essa escolha é heurística.

De acordo com a especificação proposta pelos pesquisadores esta interação do usuário para escolher outro template deve acontecer em qualquer momento após o término do workflow padrão. Por isso podemos modelar este problema com dois workflows, que escolhemos chamar *workflow principal*, que é o que existe hoje, e *workflow de refinamento*. O workflow de refinamento começa com uma interação do usuário para escolher outro(s) template(s) e prossegue tal como o principal.

Este requisito é um exemplo prático do desafio de reuso de dados mantendo proveniência, uma vez que é preciso registrar que uma determinada execução do workflow de refinamento utilizou uma saída específica de uma execução do workflow principal.

O framework proposto em [Koop et al., 2010b], a ser implementado em uma versão futura do Vistrails, prepara o caminho para tratar esse desafio. A proposta é gravar metadados de identificação de cada produto gerado em banco de dados. Entretanto o framework ainda não permite interligar um produto a uma execução específica [Koop, 2011a].

No Kepler, o esquema de dados atual permitiria que o usuário pudesse consultar quais os dados gerados em cada execução de workflow e escolher algum resultado de execução como entrada de um novo processo. O problema do esquema é que não há como rastrear qual foi a execução que interessou ao usuário, uma vez que várias execuções distintas podem gerar o mesmo dado, visto que o mesmo é identificado pelo seu código hash MD5.

Um dos objetivos deste trabalho é propor uma forma de modelagem de dados de proveniência que permita este reuso de produtos intermediários ou finais mantendo a proveniência da origem do dado, que neste caso é uma execução específica de um workflow.



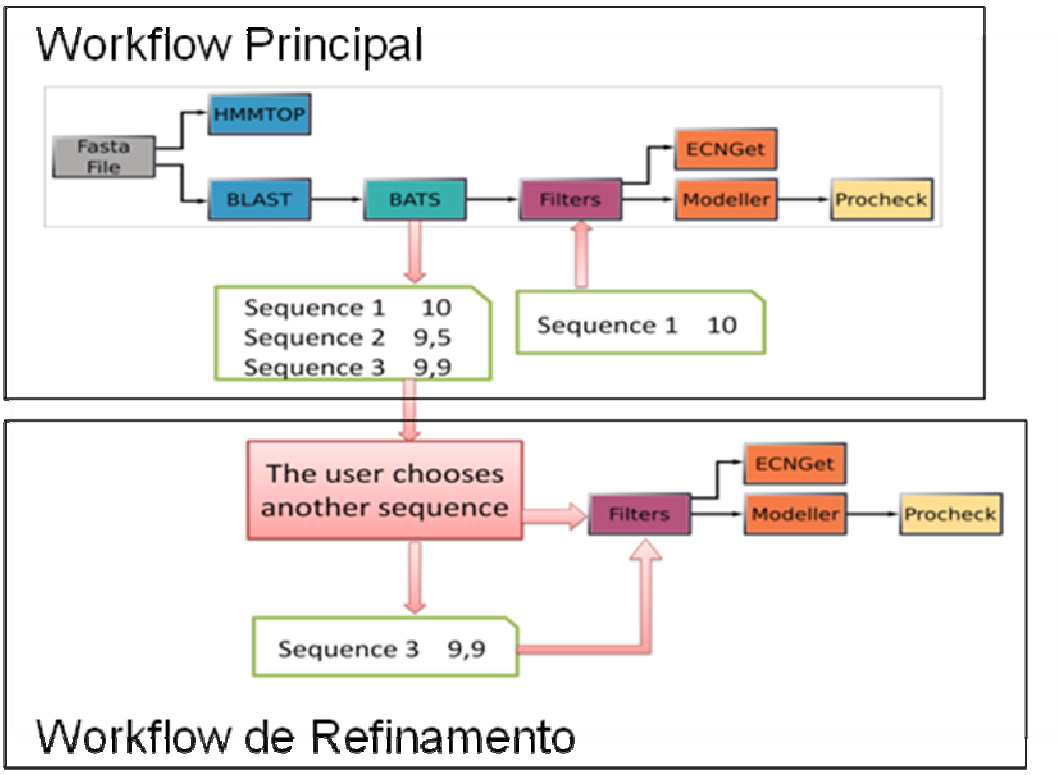


Figura 12 – Workflows principal e de refinamento

### 2.2.2.4. Descrição e Gerência de Atividades

As atividades disponibilizadas nos SGWC possuem informações como o tipo da atividade (por exemplo, alinhamento de sequências, clusterização), a forma como a mesma deve ser invocada, os parâmetros e tipos esperados, a versão, dentre outros metadados. Chamamos este conjunto de metadados de *descrição de atividade*.

Apresentaremos aqui alguns problemas em relação à descrição de atividades que acessam programas de linha de comando, uma vez que web services possuem já uma descrição própria, o que necessitaria de outras considerações.

Nos modelos de proveniência estudados são registradas poucas informações sobre as atividades utilizadas em um workflow, sejam elas utilizadas para acessar programas de linha de comando ou não. Em geral se registra apenas um identificador da atividade. Para que o usuário consulte a descrição de uma atividade existem duas alternativas: verificar via interface do sistema ou abrir o próprio arquivo da atividade, que pode ser de difícil compreensão. Ambos os casos necessitam que a atividade ainda esteja disponível no sistema, sendo que em alguns casos pode ser necessário acessar a mesma versão, o que torna o problema mais complexo. Dessa forma a reprodutibilidade da definição do workflow fica prejudicada, pois os usuários podem ter dificuldades na identificação dos programas utilizados.

Vejamos um exemplo no sistema Vistrails. Para incluir o BLAST, um programa de linha de comando muito utilizado em Bioinformática, foi criado um módulo Python de acordo com as orientações dadas no guia do usuário. A Figura 13 mostra a parte de definição de portas do módulo, na qual o usuário informa os nomes e tipos de dado esperados nas portas. Não é possível complementar a definição informando tipos de dados específicos, por exemplo, tipos de arquivos biológicos. Dessa forma sabemos apenas que o programa deve receber um arquivo em sua porta *input*, mas não temos nenhuma indicação sobre o tipo do arquivo esperado. Falta também suporte ao registro de outras informações sobre o programa, como a instituição provedora, a versão e o objetivo do mesmo.

```
reg.add_input_port(BlastLuciana, "program",
                   (core.modules.basic_modules.String, 'program'))
reg.add_input_port(BlastLuciana, "database",
                   (core.modules.basic_modules.String, 'database'))
reg.add_input_port(BlastLuciana, "input",
                   (core.modules.basic_modules.File, 'input'))
reg.add_output_port(BlastLuciana, "output",
                    (core.modules.basic_modules.File, 'output'))
```

Figura 13 – Definição das portas em módulo Vistrails de acesso ao BLAST

Na Figura 14 temos o módulo BLAST instanciado como um passo em um workflow simples. No lado direito vemos os parâmetros e tipos de dados esperados, e a parametrização do passo definida pelo usuário.

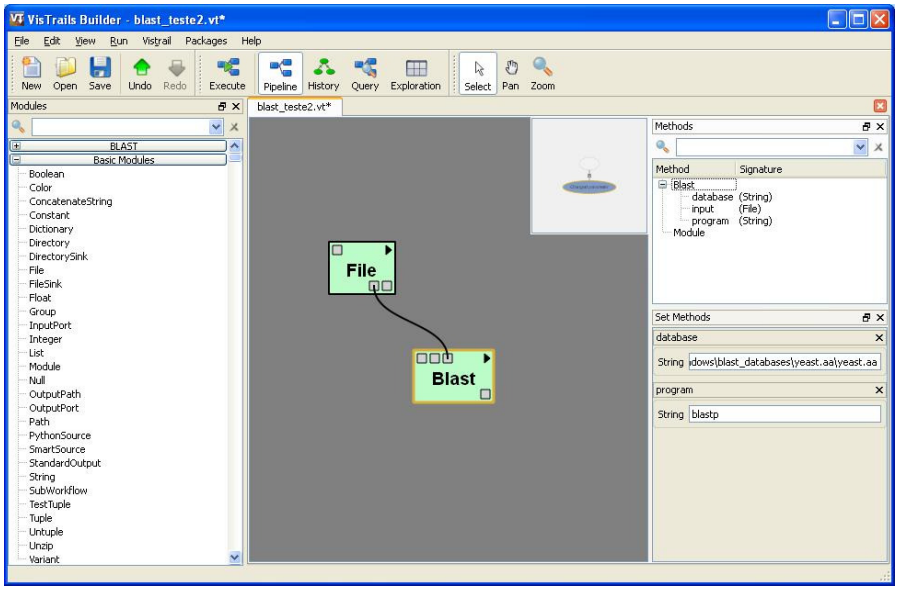


Figura 14 – Workflow que submete um arquivo local ao BLAST

Na Figura 15 mostramos como o workflow é exibido quando o módulo Blast não está disponível no sistema. O sistema informa o usuário colorindo o passo de vermelho, e as portas utilizadas no fluxo passam a ser círculos ao invés de quadrados. Ao lado direito já não se pode ver todos os parâmetros da definição da atividade, mas apenas os que foram parametrizados pelo usuário.

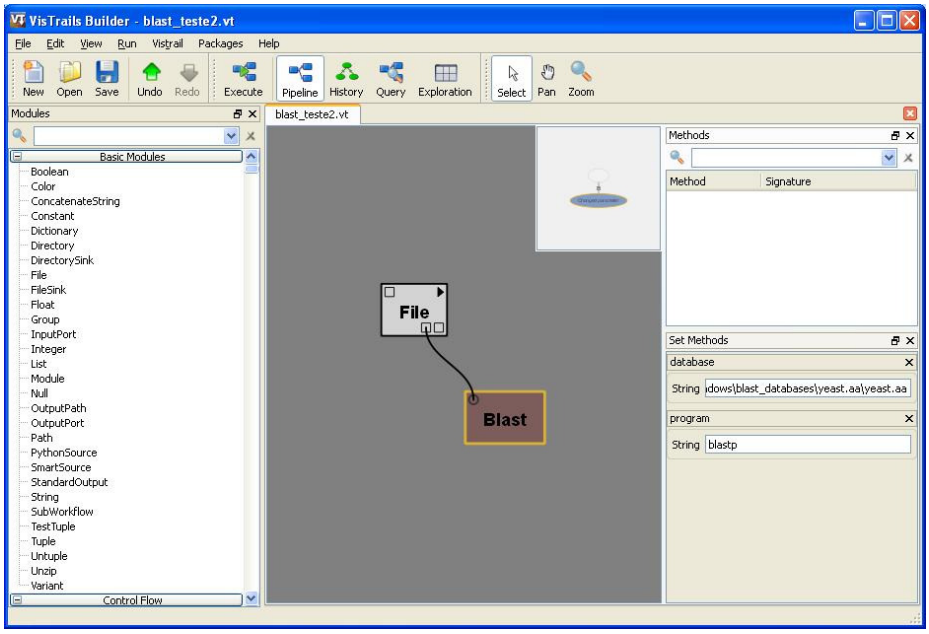


Figura 15 – Erro na execução gerado pela inexistência do módulo Blast

Este cenário leva o usuário a inferir qual foi o programa utilizado pelo nome do módulo e pela parametrização disponível. Neste caso como o BLAST é um programa muito utilizado em Bioinformática a identificação se torna mais fácil, porém continua sendo impossível saber a versão do BLAST usada, e se foi o BLAST fornecido pelo NCBI [NCBI, 2011] ou outra distribuição.

Outra característica importante em relação às atividades é a necessidade de um gerenciamento de configuração. As atividades disponibilizadas por um SGWC podem necessitar de atualizações, seja para correção do código, seja para alterações mais simples como mudança do nome de um parâmetro, por exemplo. Note-se que aqui estamos falando de alterações na atividade propriamente dita, e não no passo do workflow.

A necessidade de um gerenciamento de configuração de atividades é citada em [Mattoso et al., 2010]. Esse tipo de gerenciamento permite que sejam geradas versões de atividades de acordo com as atualizações realizadas, assim como a comparação entre versões diferentes.

Ressaltamos que atividades que acessam programas de linha de comando possuem dois níveis de descrição, um para o código da atividade que encapsula a chamada ao programa, e outro para o próprio programa. Ainda no Vistrails, temos um código Python, que é o código da atividade, que por sua vez irá invocar o programa BLAST, ou qualquer outro. O módulo deve ser versionado independentemente à versão do programa. Suponha que o usuário definiu um módulo *Blast* simplificado que considera apenas os parâmetros *program*,

*database*, *input*, que são os que ele mais utiliza. Este módulo pode ser definido como versão 1.0 e invocar uma versão do BLAST 2.2.22. Agora suponha que o usuário deseja adicionar o parâmetro *e-value*, disponibilizando-o como porta, para todos os seus futuros workflows, ou mesmo para os já criados. O módulo receberá uma versão 1.1, por exemplo, mas o programa invocado continua o mesmo.

Consideramos importante separar o contexto de descrição do programa do contexto de descrição da atividade, o que não encontramos nos sistemas estudados.

#### 2.2.2.5. Consultas sobre Grafos

Em [Holland et al., 2008] é apresentado um dos problemas para a escolha de um modelo de proveniência: Das informações de proveniência que um sistema armazena derivam informações sobre relacionamentos entre objetos, que formam por sua vez um grafo dirigido. Consultas sobre a proveniência são baseadas em caminhamento sobre um determinado grafo. Os modelos mais utilizados hoje apresentam problemas para esse tipo de informação. O modelo relacional não suporta grafos nativamente, o XML não representa de forma nativa elementos com múltiplos parentes, e o RDF unido ao SPARQL para consultas possui várias restrições em relação a consultas sobre caminhos de um grafo.

#### 2.2.2.6. Facilidade de Interação

O cenário de utilização de um sistema de workflow de negócios é bastante diferente do cenário de uso de um sistema de workflow científico. As empresas em geral provêem treinamentos para que os funcionários estejam capacitados a construir workflows, testá-los, e disponibilizá-los para uso. Os workflows são construídos por profissionais de informática, que recebem uma especificação do processo de negócio em algum nível de abstração e são responsáveis por representar o processo na ferramenta de workflow.

Ao tratarmos de workflows científicos, o cenário é de acúmulo das tarefas de modelagem e construção por um profissional que tem, em geral, um conhecimento limitado de informática. Por isso, para que um sistema se torne útil para um biólogo, por exemplo, é necessário que seja simples e intuitivo. É

fundamental que as funcionalidades da ferramenta sejam oferecidas de forma adequada a este tipo de usuário não-especialista.

Vários trabalhos [Davidson e Freire, 2008; McPhillips et al., 2009] sobre proveniência em SGWC citam a questão da necessidade de facilitar a interação com o usuário, seja em relação à construção dos workflows, seja em relação a realizar consultas sobre os dados de proveniência. Uma das formas de facilitar consultas à base de proveniência é criar consultas pré-definidas descritas em linguagem natural [Marinho et al., 2010]. De fato, um usuário de Bioinformática dificilmente irá direcionar esforços para aprender uma linguagem de consulta como o XQuery ou o Lorel, usado no sistema PASS [Holland et al., 2008].

#### 2.2.2.7. Interoperabilidade

A interoperabilidade foi tema central do 2º Provenance Challenge. Ao criar uma representação para seus workflows, os desenvolvedores de um sistema devem se perguntar se será possível exportar as informações de proveniência para que possam ser compreendidas por outro sistema. Isso se tornou um grande desafio para os SGWC devido à heterogeneidade de modelos existentes. Alguns sistemas, por exemplo, Taverna e Vistrails, implementaram uma função de exportação de dados para o OPM como uma forma de promover interoperabilidade.

Em [Marinho et al., 2010] é proposta uma solução para a interoperabilidade entre sistemas através de um componente externo de proveniência. Nesta proposta o cenário de interesse é a necessidade de juntar workflows criados em diferentes sistemas em apenas um experimento, permitindo que o novo workflow possa ser executado automaticamente. O software desenvolvido (ProvManager) captura dados de proveniência através da instrumentalização dos workflows com adição de passos que invocam as chamadas atividades de proveniência, responsáveis por enviar ao ProvManager os dados capturados na execução. Esta instrumentalização pode ser realizada por enquanto no Kepler e no Vistrails, permitindo portanto a interoperabilidade entre esses dois sistemas.

#### 2.2.2.8. Modelagem Conceitual

A maioria dos sistemas não suporta a composição de um experimento em mais alto nível, definindo algoritmos ao invés de implementações específicas,

que são os componentes. Por exemplo, em um experimento de Bioinformática, o usuário poderia especificar que em determinado passo deve-se realizar uma tarefa de alinhamento local de sequências, sendo que ele ainda não sabe qual implementação irá utilizar dentre as existentes, por exemplo BLAST ou Smith-Waterman.

Em [Mattoso et al., 2010] se defende que os SGWC deveriam suportar esse nível de abstração para se aproximar da forma de pensar do pesquisador. Nessa proposta, a composição do experimento científico incluiria a definição em alto nível das tarefas, e a sua instanciação concreta em componentes específicos.

Dentre os SGWC estudados, o sistema REDUX [Barga e Digiampietri, 2008], participante do First Provenance Challenge, possui um modelo relacional em camadas que permite a representação de workflows em quatro níveis. No nível chamado L0, o workflow é uma composição de atividades abstratas. O nível L1 é uma instanciação do nível L0, apresentando as implementações escolhidas pelo usuário para cada atividade abstrata. Já os níveis L2 e L3 representam a execução do experimento, sendo que no nível L2 as informações de interesse são os dados de input e parâmetros de cada tarefa, caminhos escolhidos, atividades adicionadas em tempo de execução e parâmetros alterados. O nível L3 informa detalhes da execução, como códigos de saída, arquivos intermediários, duração das tarefas e outros.

Uma ferramenta complementar aos SGWC, chamada GexpLine [Oliveira et al., 2010], provê suporte a essa abordagem e permite a modelagem abstrata das chamadas linhas de experimento, definidas no mesmo trabalho como workflows conceituais capazes de gerar diversas instâncias de workflows concretos, ou seja, aptos para execução. Os workflows concretos são gerados para execução em um SGWC existente, para o qual deve ser implementado um *cartridge*, que efetua a conversão do nível abstrato para o nível concreto do sistema.

Dentre as vantagens apontadas pelos trabalhos citados estão, por exemplo, a possibilidade de compor um experimento de forma conceitual e relacioná-lo a diversas instanciações, que são as várias tentativas de solução realizadas pelo cientista, e a possibilidade de compartilhar apenas partes do experimento de acordo com o interesse do usuário.

### 2.2.2.9. Atualização de Workflow

A captura da proveniência dos componentes utilizados permite que workflows que acessam versões diferentes das disponíveis no sistema sejam identificados, e possivelmente atualizados manualmente ou mesmo automaticamente. Suponha que um usuário compartilha seu workflow, que usa um componente cuja versão é a 2.0, com outro usuário. Na configuração do sistema do segundo usuário não é disponibilizada a versão 2.0, mas sim uma versão anterior ou posterior. Ao abrir a definição do workflow, o sistema pode alertar o usuário e permitir a atualização manual ou mesmo automática do workflow para que seja acessada a versão disponível no sistema.

O problema de tratar atualizações de workflows que utilizam versões diferentes da disponível no sistema não tem sido explorado pelos SGWC existentes, exceto Vistrails que elaborou um modelo de tratamento para o problema e o descreve em [Koop et al., 2010a]. Note-se que este modelo não permite que o componente seja atualizado para qualquer outra versão, mas sim para a versão disponível na configuração do sistema.

### 2.2.2.10. Modelo de Proveniência

Os SGWC adotam em geral um modelo ad-hoc para representar um workflow e suas execuções. Alguns sistemas como Kepler e Taverna armazenavam as definições dos workflows apenas em arquivos XML de schemas específicos, porém com a crescente preocupação com proveniência de dados esses sistemas produziram esquemas baseados no modelo relacional para representar toda ou parte da informação de proveniência. A escolha de uma forma de modelagem e de uma forma de armazenamento é um desafio devido a todas as questões anteriormente levantadas.

Um dos problemas identificados neste trabalho foi o uso de modelos pouco genéricos e muito dependentes de tecnologia por alguns dos sistemas estudados (seção 2.1.3.3), o que dificulta sua compreensão e a interoperabilidade com outros sistemas.

### 2.2.3.
### Níveis de Reprodutibilidade

Os dados e metadados registrados por um sistema que suporta proveniência permitem diferentes graus de reprodutibilidade. Estas informações são relativas à especificação do workflow, aos dados consumidos e produzidos, às atividades utilizadas, e aos acessos a programas externos ao sistema.

É proposta nesta dissertação uma classificação em pelo menos dois níveis de reprodutibilidade como os principais a serem considerados em um SGWC: o *reuso da definição* e a *reprodutibilidade estrita*. Em cada nível descrevemos alguns dados e metadados que podem ser registrados em relação a esses níveis de reprodutibilidade.

#### 2.2.3.1.
#### Reuso de Definição

Um sistema suporta *reuso de definição* se para um determinado workflow há possibilidade de acessar a especificação do workflow. Este tipo de reprodutibilidade está relacionado ao armazenamento de dados de proveniência prospectiva, realizada por todos os sistemas estudados neste trabalho em diferentes graus de detalhamento.

Neste nível o interesse está na especificação do processo, não em produzir os mesmos dados nem em garantir o uso das mesmas atividades. Trata-se de descrever quais os passos do workflow, as conexões entre eles, parâmetros utilizados, e outros dados necessários e suficientes para reutilizar esta definição de workflow.

Quanto maior o nível de detalhe fornecido em uma especificação de workflow mais fácil será sua compreensão e reuso. Por exemplo, em relação aos dados de entrada de um workflow, uma especificação em geral não contém os dados em si, mas pode conter referências para os dados. Estas referências podem variar desde apenas um nome de arquivo (ex. swissprot.fasta) até descrições sobre o tipo ou a procedência do dado [Guimarães, 2009]. Informações sobre o tipo e a origem do dado permitem um maior entendimento do objetivo do workflow.

Já as atividades de um workflow podem ter descrições tal como abordado na seção 2.2.2.4. Estas descrições facilitam a compreensão e o reuso da definição de um workflow. Se um usuário quer reutilizar uma especificação mas não tem acesso às atividades e acessos externos utilizados ele poderá procurá-

los através dos metadados disponíveis sobre estes. Estes metadados facilitam também a identificação de uma atividade similar, caso a própria não seja encontrada.

### 2.2.3.2. Reprodutibilidade Estrita

Nomeamos *reprodutibilidade estrita* a capacidade de um sistema de reproduzir uma execução de workflow com os mesmos dados e atividades conforme existiam no momento da execução. Para tal o sistema deve registrar a definição de workflow conforme visto na seção anterior, e garantir que os dados ainda possam ser acessados no seu conteúdo original e que as atividades utilizadas no momento da execução ainda estejam disponíveis.

Com relação aos dados, se estes são acessados diretamente em uma máquina do ambiente do usuário, a solução trivial é armazenar uma cópia dos dados consumidos e produzidos em cada execução. O problema surge quando são utilizados arquivos de grande volume de dados.

Entretanto, em apenas alguns casos o usuário irá necessitar reproduzir o workflow com os dados de entrada originais, como para garantir a confiabilidade de uma publicação científica. Por isso, defendemos que um SGWC deve prover a reprodutibilidade estrita como uma funcionalidade a ser habilitada ou desabilitada conforme interesse do usuário.

Uma funcionalidade que pode ser oferecida neste contexto é identificar quando **não** é possível reproduzir estritamente o workflow. Esta informação depende da identificação se o dado original foi alterado, sem o compromisso de efetivamente armazenar o mesmo. Isto é possível através da utilização de uma função de *hash* sobre o dado original, como um cálculo MD5 [Rivest, 1992] do mesmo. Este cálculo resulta em uma cadeia de caracteres gerada pelo algoritmo de *hash* MD5, que é utilizada em muitos sistemas como um identificador único do dado, dada sua baixíssima probabilidade de colisões. Ao armazenar o MD5 do arquivo original em cada execução passa a ser possível identificar se o arquivo que tenho hoje na minha instalação corresponde ao mesmo que tinha no momento de uma determinada execução. Dessa forma, pode-se alertar o usuário que não é possível reproduzir de forma estrita a execução com esse mesmo arquivo, já que houve uma alteração.

Com relação às atividades utilizadas, para que seja possível executar as atividades nas mesmas versões usadas em uma determinada execução o sistema deve garantir o acesso a essas versões.

No Vistrails por exemplo, na versão 1.6, o sistema possui limitações em relação a reprodutibilidade estrita pois disponibiliza apenas uma versão de cada pacote de módulos. Para garantir a reprodutibilidade de um workflow que utiliza versões de módulos diferentes das que estão habilitadas no sistema o usuário deve efetuar um backup manual das versões ou utilizar um sistema de controle de versões. Até o momento da elaboração desta dissertação o Vistrails não efetua nenhum tipo de gerenciamento de versões dos seus pacotes, porém este tipo de gerenciamento pode vir a ser integrado em alguma versão futura do sistema [Koop, 2011b].

Para garantir acesso a diferentes versões de atividades, estas poderiam ser versionadas por um sistema de controle de versões que fosse integrado ao SGWC, como sugerido em [Koop, 2011a], porém nenhum dos sistemas estudados oferece essa funcionalidade. Entretanto podem ser empregados métodos mais simples como efetuar uma cópia do código da atividade (seja um código fonte, seja um arquivo em XML) sempre que a mesma é alterada e criar uma versão nova nas alterações, mantendo a ligação entre o passo do workflow e a atividade executada através da versão da mesma.

Da mesma forma que gravamos o valor de hash MD5 dos dados, pode-se também gravá-lo para cada atividade. Essa é uma forma simples de responder a pergunta “Essa atividade ainda é a mesma que o usuário utilizou naquela execução?”.

Em relação aos acessos externos, a reprodução estrita se torna mais desafiadora pela falta de controle do sistema de gerência sobre o comportamento do recurso externo. Suponhamos uma atividade que execute um script criado pelo usuário, como o BATS do workflow MHOLline. Mesmo que se faça um backup do script que é invocado, não se garante a reprodução estrita visto que o programa pode acessar bibliotecas locais e mesmo variáveis de ambiente que o sistema desconhece.

Entretanto, o backup do programa invocado ou o registro de um código *hash* do mesmo podem ser alternativas simples e úteis para o usuário. Se não houver dependências, a cópia do script será suficiente para reproduzir o programa. Se houver dependências, pelo menos irá garantir a reprodutibilidade até o nível do próprio programa. O código MD5 permite comparar o arquivo existente atualmente na configuração do usuário com o que foi executado.

No caso do acesso a web services o problema é ainda mais complexo, pois não se tem acesso direto a instalação efetiva do programa na instituição provedora do serviço.

## 2.3. Objetivos

Na seção anterior, listamos alguns desafios de proveniência, dos quais escolhemos alguns que possuem mais enfoque na área de Banco de Dados para servirem como motivação para esse trabalho. Listamos na Tabela 1 estes desafios, relacionando-os com alguns objetivos da presente dissertação.

Tabela 1 – Objetivos do Projeto de Proveniência

| Desafio | Objetivos |
|---|---|
| Modelo de Proveniência | • Buscar uma forma de modelagem genérica e independente de tecnologia<br>• Utilizar o OPM |
| Descrição e Gerência de Atividades | • Incluir descrição de atividades na modelagem de proveniência, considerando descrição de programas de linha de comando.<br>• Propor uma forma de gerenciar atividades. |
| Gerência de Dados Consumidos e Produzidos | • Propor uma solução de gerenciamento de dados consumidos e produzidos. |
| Reuso de Dados | • Propor uma forma de reutilizar um dado produzido como entrada em um workflow mantendo a proveniência da origem. |
| Reprodutibilidade | • Permitir a reprodução de uma execução de workflow. |

O projeto foi desenvolvido considerando um cenário específico, no qual os workflows considerados são estritamente do tipo data-flow, que para o contexto deste trabalho definimos como um workflow cujas ligações entre passos são dependências de dados. Estamos portanto desconsiderando fluxos de controle.

Note-se que este cenário limita o projeto a um determinado escopo de workflows, que era o escopo do nosso interesse, cuja motivação foi apresentada na seção Workflows em Bioinformática.