Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Gerenciamento de Dados de Proveniência de Workflow de Bioinformática com Banco de Dados Baseados em Grafo

Rodrigo Pinheiro de Almeida

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientadora
Prof.ª Dr.ª Maristela Terto de Holanda

Brasília
2015

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Informática

Coordenador: Prof.ª Dr.ª Alba Cristina M. de Melo

Banca examinadora composta por:

Prof.ª Dr.ª Maristela Terto de Holanda (Orientadora) — CIC/UnB
Prof.ª Dr.ª Célia Ghedini Ralha — CIC/UnB
Prof. Dr. Angelo Roncalli Alencar Brayner — CIC/UnB

CIP — Catalogação Internacional na Publicação

Almeida, Rodrigo Pinheiro de.

Gerenciamento de Dados de Proveniência de Workflow de Bioinformática com Banco de Dados Baseados em Grafo / Rodrigo Pinheiro de Almeida. Brasília : UnB, 2015.
139 p. : il. ; 29,5 cm.

Dissertação (Mestrado) — Universidade de Brasília, Brasília, 2015.

1. Proveniência de dados, 2. Banco de dados de Grafo, 3. NoSql,
4. Neo4J, 5. PROV-DM.

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Gerenciamento de Dados de Proveniência de Workflow de Bioinformática com Banco de Dados Baseados em Grafo

Rodrigo Pinheiro de Almeida

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Prof.ª Dr.ª Maristela Terto de Holanda (Orientadora)
CIC/UnB

Prof.ª Dr.ª Célia Ghedini Ralha
CIC/UnB

Prof. Dr. Angelo Roncalli Alencar Brayner
CIC/UnB

Prof.ª Dr.ª Alba Cristina M. de Melo
Coordenador do Mestrado em Informática

Brasília, 26 de Junho de 2015

Agradecimentos

Primeiramento a Deus, pela saúde e força de vontade. A minha esposa Thayres que esteve sempre ao meu lado, me estimulando e encorajando nos momentos de fraqueza. A minha orientadora Prof.ª Dr.ª Maristela Terto de Holanda pelas sábias orientações, ajuda inestimável, paciência e principalmente pela oportunidade dada. E em especial aos meus pais e minha tia Maria de Lurdes, que sem eles nada disso seria possível.

Resumo

Muitos experimentos científicos na bioinformática são executados como *workflows* computacionais. Algumas vezes para a validação e reconhecimento de um experimento é necessário reexecutá-lo sob as mesmas circustâncias nas quais foi originado. Proveniência de dados diz respeito à origem ou procedência dos dados. Para facilitar o entendimento e análise dos resultados, é interessante ter os dados de proveniência, que detalham e documentam a história e os caminhos dos dados de entrada, do início do experimento até o final. Portanto, neste contexto, a proveniência de dados pode ser aplicada para realizar a rastreabilidade de um experimento. Considerando que uma execução de um workflow pode ser representado por um grafo, como definido no modelo PROV-DM, este documento apresenta uma arquitetura capaz de realizar a proveniência de dados de experimentos científicos na bioinformática de forma automática e armazenamento em bancos de grafo.

Palavras-chave: Proveniência de dados, Banco de dados de Grafo, NoSql, Neo4J, PROV-DM.

Abstract

Many scientific experiments in bioinformatics are executed as computational workflows. Sometimes for the validation and recognition of an experiment is need rerun under same circumstances in which it was originated. Date provenance concers the origin data. To facilate the understanding and analysis of the results is interesting to have the source data, detailing and documeting the history and the paths of the input data, the beginning of the experiment until the end. Therefore, in this context, the data provenance can be applied to realize an experiment traceability. Whereas an execution of a workflow can be represented by a graph, as defined in PROV-DM model, this document presents an architecture able to perform the data provenance of iscientif experiments in bioinformatics automatically and storage graph database.

VizJS (2015), Huacarpuma et al. (2011), Frishman and Valencia (2009), Stein (2001), Alvarez (2009)

Keywords: Data Provenance, Graph database, Neo4J, PROV-DM.

Sumário

1 Introdução **1**
- 1.1 Contextualização . . . 1
- 1.2 Problema . . . 2
- 1.3 Objetivos . . . 2
  - 1.3.1 Objetivos Específicos . . . 2
- 1.4 Estrutura do Trabalho . . . 3

2 Fundamentação Teórica **4**
- 2.1 Uma Breve História . . . 4
- 2.2 *Workflow* de Bioinformática . . . 5
- 2.3 Conceito de Proveniência de Dados . . . 8
  - 2.3.1 Modelos de Proveniência de Dados . . . 9
  - 2.3.2 PROV-DM - *Provenance Data Model* . . . 10
  - 2.3.3 PROV-DM em *Workflow* de Bioinformática . . . 13
- 2.4 Modelo de Dados de Grafo . . . 15
- 2.5 Banco de dados de Grafo em *NoSQL* . . . 21
- 2.6 Banco de Dados Relacional *Versus* Grafo . . . 22
- 2.7 Trabalhos Correlatos . . . 25

3 Arquitetura Proposta **29**
- 3.1 Contextualização da Proposta . . . 29
- 3.2 Modelo de Dados de Proveniência em Bancos de Dados de Grafo . . . 30
- 3.3 Arquitetura Proposta . . . 31
  - 3.3.1 MVC - *Model-View-Controller* . . . 31
  - 3.3.2 Definição dos módulos . . . 33
  - 3.3.3 Interface Web . . . 34
  - 3.3.4 *Controller* . . . 35
  - 3.3.5 Módulo de Processamento . . . 36
  - 3.3.6 Módulo de Armazenamento . . . 37
  - 3.3.7 Fila . . . 38
  - 3.3.8 Watcher . . . 39
- 3.4 Bancos de Dados de Grafo - *Neo4J* . . . 40

4 Estudos de Caso **41**
- 4.1 Simulador *Provenance* . . . 41
  - 4.1.1 Ambiente Computacional . . . 41
  - 4.1.2 Bibliotecas Externas . . . 42

4.1.3 Banco de Dados de Grafo - *Neo4J* 43
4.1.4 Visão Geral de Um Grafo de Proveniência 45
4.2 Estudo de Caso 1 - *Alpha-amylase* 46
4.3 Estudo de Caso 2 - Rim-Fígado 48

5 Conclusão **55**
5.1 Trabalhos Futuros 56

Referências **57**

Lista de Figuras

2.1 Dupla hélice de DNA (Watson et al., 1953) 5
2.2 Exemplo de *workflow* para projetos genoma e transcritoma. 6
2.3 Exemplo do processo de mapeamento. 6
2.4 Exemplo do processo de montagem. 7
2.5 Exemplo do processo de transcrição (A) e do problema de alinhamento com *splice junctions* (B). 7
2.6 Representação gráfica dos nós do modelo PROV-DM (W3C, 2015). 11
2.7 Representação gráfica das relações do modelo PROV-DM (W3C, 2015). . . 12
2.8 Exemplo de grafo do modelo PROV-DM adaptado do (W3C, 2015). 14
2.9 Exemplo de grafo. 15
2.10 Exemplo de grafo de atributo. 18
2.11 Exemplo de grafo simples. 19
2.12 Exemplo de multigrafo. 19
2.13 Exemplo de hipergrafo. 20
2.14 Exemplo de grafo aninhado. 20
2.15 *Schema* relacional do modelo PROV-DM. 24

3.1 Mapeamento das entidades PROV-DM para o modelo de grafo de atributos. 32
3.2 Um projeto agrupa diferentes experimentos executados. 32
3.3 Objetos utilizados no MVC e suas interações. 33
3.4 Arquitetura Proposta. 34
3.5 Telas de login e cadastro de usuários. 35
3.6 Tela principal do sistema. 36
3.7 Diagrama de serviços da camada de processamento. 37
3.8 Diagrama de serviços da camada de processamento. 37
3.9 Diagrama de serviços da camada de armazenamento. 38
3.10 Funcionamento de uma fila. 39
3.11 Funcionamento do módulo *Watcher* 39

4.1 Exemplo de um grafo construído a partir da linguagem DOT. 43
4.2 Tela principal do simulador de proveniência. 45
4.3 Workflow resumido do experimento com a bactéria Alpha-amylase indicando os programas utilizados. 46
4.4 Grafo de proveniência do experimento da bactéria *Alpha-amylase*. 47
4.5 Tela de detalhes da entidade *Account* do *workflow Alpha-amylase*. 48
4.6 Grafo de proveniência do grupo da Família_57 49
4.7 *Workflow* resumido do experimento indicando os programas utilizados. . . 49

4.8 Tela da Atividade A001_Filtro_Rim. 51
4.9 Grafo personalizado da fase de Filtragem. 52
4.10 Parte I - Grafo do experimento que trata a expressão diferencial entre células de rim e fígado. 53
4.11 Parte II - Grafo do experimento que trata a expressão diferencial entre células de rim e fígado. 54

Capítulo 1

Introdução

1.1 Contextualização

Projetos de Bioinformática consistem na execução de diversos experimentos com sequencias obtidas por sequenciadores de alto desempenho tais como *Illumina* (Bentley, 2006) ou 454 *Roche* (Rothberg and Leamon, 2008). Estes equipamentos são capazes de gerar, em poucas horas, milhões de fragmentos de código que precisam ser analisados (Schuster, 2008). Esses milhões de fragmentos podem gerar terabytes de dados, que são armazenados em diferentes arquivos com diversos formatos e envolvem a utilização de distintas ferramentas computacionais cujas configurações e parâmetros de entrada podem afetar fortemente os resultados obtidos.

O gerenciamento da execução desses experimentos é realizada normalmente através de um *workflow* científico, que é uma abstração que define as etapas de execução de um experimento e a sequência em que tais etapas ocorrem, de forma a corroborar ou refutar uma hipótese científica (Jarrard, 2001). Cada etapa do workflow envolve a execução de um ou mais programas, que são responsáveis pela transformação dos dados de entrada e a produção de dados de saída. Entretanto, para que um experimento seja válido sob o ponto de vista científico, o seu resultado deve ser passível de reprodução por terceiros, logo é importante armazenar dados tanto do ambiente de execução quanto dos experimentos propriamente ditos. Estes dados podem ser obtidos a partir de dados de proveniência, seja em ambientes centralizados ou distribuídos (Altintas et al., 2006).

A proveniência de dados visa descrever os acontecimentos e insumos utilizados na geração de uma determinada informação. Segundo Buneman et al. (2001) proveniência de dados é "... a descrição das origens de uma peça de dados e do processo pelo qual ela chegou em um banco de dados." (livre tradução), ou seja, para garantir a proveniência de dados se faz necessário guardar tanto a origem dos dados utilizados como matéria-prima, quanto os processos que transformaram estes dados no produto final.

Dessa forma, a proveniência de dados vem se tornando cada vez mais presente no ambiente científico, tanto para garantir a origem dos dados como para avaliar a sua acurácia. Porém, antes de definir quais informações são necessárias para garantir a proveniência e como serão gerenciadas, é preciso definir uma forma de organizá-las a fim de que se possa, posteriormente, recuperá-las e entendê-las de forma que possam trazem o máximo de benefícios possíveis. Com esse objetivo, encontra-se na literatura a definição de diversos modelos de proveniência de dados tais como OPM (OPM, 2015), PROV-DM (W3C,

2015), entre outros. Estes modelos tem como objetivo tornar mais clara a idéia de proveniência de dados além de criar estruturas práticas para a organização e gerenciamento dos dados de proveniência. Neste contexto surgiu o conceito de Sistemas Gerenciadores de Proveniência (SGP) (Moreau et al., 2008) voltados ao armazenamento, recuperação e gerenciamento de dados de proveniência relacionadas aos experimentos executados.

Um dos desafios de um SGP para o gerenciamento de proveniência de dados em projetos de bioinformática e a diversidade de programas utilizados gera diferentes tipos de resultados, dependendo do tipo de análise que está sendo realizada no projeto. Por isso, realizar a captura de forma automática, armazenar e gerenciar essas informações, em formatos diversos, depende de uma prévia padronização que seja flexível o suficiente para aceitar a grande maioria desses processos e formal o suficiente para permitir um gerenciamento adequado.

Neste trabalho propõe-se uma arquitetura capaz de realizar a captura de forma automática dos dados de proveniência e um modelo de armazenamento baseado em grafos extendido a partir do modelo PROV-DM. Para demonstrar a viabilidade e corretude da arquitetura foi implementado o simulador *Provenance*, testado com informações reais de projetos de bioinformática executados no Laboratório de Biologia Molecular (BIOMOL) do Departamento de Biologia Celular (CEL) da Universidade de Brasília (UnB). Os aspectos considerados para o desenvolvimento do simulador foi a simplicidade de uso e a independência de ferramentas de Sistemas Gerenciadores de *Workflow*.

Portanto, no contexto de *workflows* da bioinformática, este trabalho apresenta uma arquitetura para recuperação de proveniência de dados de maneira automatizada, usando como modelo de dados o PROV-DM e o armazenamento em bancos de dados em grafo.

1.2 Problema

Projetos de bioinformática são suportados por workflows que podem ser executados várias vezes com parâmetros diferentes. O resultado de cada execução é fortemente afetada tanto pelos dados quanto pelos programas e parâmetros utilizados. As saídas de cada passo do *workflow* não são registradas e não são mantidos históricos dessas execuções, dificultando a análise e reprodução do experimento.

1.3 Objetivos

O objetivo geral deste trabalho é a definição de uma arquitetura de captura de proveniência de dados automatizada no contexto de *workflow* em Bioinformática, utilizando o modelo de proveniência PROV-DM e armazenamento em bancos de dados em grafo.

1.3.1 Objetivos Específicos

No intuito de atingir o objetivo geral, foram definidos alguns objetivos específicos:

- Implementar uma arquitetura de captura automática de proveniência de dados em *workflow* de Bioinformática, utilizando bancos de dados de grafos;

- Definir o modelo de dados de proveniência de *workflow* de Bioinformática baseada em grafo;
- Realizar estudo de caso com *workflows* científicos reais da Bioinformática para validação da arquitetura proposta;
- Avaliar os resultados obtidos.

1.4 Estrutura do Trabalho

Este documento está estruturado nos capítulos a seguir:

- O Capítulo 2 apresenta o referencial teórico necessário para o desenvolvimento dessa pesquisa, tais como projeto de bioinformática, a proveniência de dados, o modelo PROV-DM, o armazenamento de dados baseados em *NoSQL* e os trabalhos relacionados;
- O Capítulo 3 especifica a proposta de arquitetura de captura automática para a proveniência de dados no contexto de *workflow* da Bioinformática com banco de dados baseados em grafo;
- O Capítulo 4 apresenta os estudos de casos aplicados para validação da arquitetura;
- E por fim no Capítulo 5, são apresentadas as conclusões e trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo são tratados os conceitos relacionados a *workflows* de Bioinformática, proveniência de dados e bancos de dados *NoSQL* baseado em grafo, sendo dividido nas seguintes seções: a Seção 2.1 revisa o surgimento dos estudos genômicos. Seção 2.2, os conceitos relacionados aos projetos de Bioinformática; Seção 2.3 apresenta a definição e os modelos de proveniência assim como a sua aplicação em Bioinformática; Seção 2.4, modelos de dados de grafo, e por fim a Seção 2.5, alguns trabalhos na literatura sobre proveniência de dados.

2.1 Uma Breve História

As pesquisas com DNA iniciaram na década de 1860, quando o bioquímico alemão Johann Friedrich Miescher (Dahm, 2008) descobriu compostos de natureza ácida no núcleo das células e deu o nome de nucleína. Estes compostos são conhecidos como bases hidrogenadas e são os principais elementos que compõe o DNA. Mais de 60 anos depois, em 1929, o bioquímico lituano Phoebus Aaron Theodore Levene publicou o artigo *"The Structure of Thymonucleic Acid"* (Levene and London, 1929), demonstrando como as quatro bases Adenina, Citosina, Guanina e Timina eram ligadas para formar o DNA.

Porém, a corrida para o mapeamento do DNA só foi iniciado a partir de 1953, quando os cientistas Watson e Crick descreveram a estrutura do DNA chamada dupla hélice (Watson et al., 1953) (Figura 2.1). Ainda se passariam 24 anos, até que o primeiro organismo tivesse seu DNA completamente mapeado, o que foi realizado em 1977 por um grupo liderado pelo cientista Frederick Sanger (Sanger et al., 1978). O organismo mapeado foi o bacteriófago Phi-X174, cujo DNA é composto por cerca de 5000 bases, relativamente curto se comparado ao humano, com 3 bilhões de bases (Setubal and Meidanis, 1997).

A partir de então, novas tecnologias tem sido desenvolvidas, diminuindo o tempo e reduzindo os custos desses mapeamentos. Em 2004 novos equipamentos chamados sequenciadores de DNA de alto desempenho chegaram ao mercado, revolucionando os projetos de mapeamento de DNA (Mardis, 2008). Entre eles pode-se citar o *454/Roche* (Rothberg and Leamon, 2008), que foi a primeira tecnologia após o Método Sanger a mapear um genoma humano, e o *Illumina Genome Analyzer* (Bentley, 2006). Esses equipamentos conseguem sequenciar milhares de fragmentos de DNA em poucas horas, além de garantir melhor qualidade em relação aos métodos anteriores.

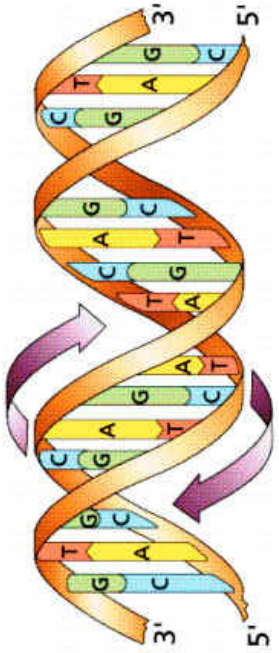

Figura 2.1: Dupla hélice de DNA (Watson et al., 1953)

2.2 *Workflow* de Bioinformática

A informática fornece suporte para diversas áreas de pesquisa tanto desenvolvendo modelos de dados como algoritmos. A bioinformática surgiu da utilização de ferramentas computacionais para as soluções de problemas da biologia tanto na necessidade de processamento como no armazenamento e recuperação de dados.

Workflows de bioinformática estão ligados diretamente ao processamento de dados biológicos e, em geral, são relacionados aos processos de sequenciamento de DNA ou RNA. Processos de sequenciamento de DNA ou RNA consistem na tarefa de descobrir, para um determinado organismo, qual é a sequência de bases que forma cada fragmento de DNA ou RNA que está sendo investigado. Assim pode-se definir como projetos genoma aqueles que consistem em pesquisar DNA genômico (cromossomos), enquanto que projetos transcritoma consistem no estudo dos transcritos (RNA). A partir desses fragmentos, tanto de DNA quanto de RNA, diversos tipos de processos computacionais podem ser executados dependendo do objetivo final do projeto.

De forma geral, projetos genoma e transcritoma, possuem suporte computacional, nos quais são projetados *workflows*, que transformam fragmentos de entrada de tal forma a extrair delas informações como funções biológicas e localização dentro da célula. A Figura 2.2 mostra um exemplo de *workflow* que apresenta um conjunto de dados obtidos a partir do sequenciamento de alto alto desempenho e serve como insumo para execução do *workflow* no qual possui três fases (filtragem, mapeamento/montagem e análise) explicadas a seguir.

Inicialmente, os biólogos realizam processos laboratoriais de coleta e replicação do material biológico. Depois esse material é enviado para o sequenciamento de pequenas porções do DNA ou RNA (dependendo do tipo do projeto), chamadas *reads*. O tamanho de cada *read* obtida depende do equipamento utilizado podendo variar de 90 à 600 bases.

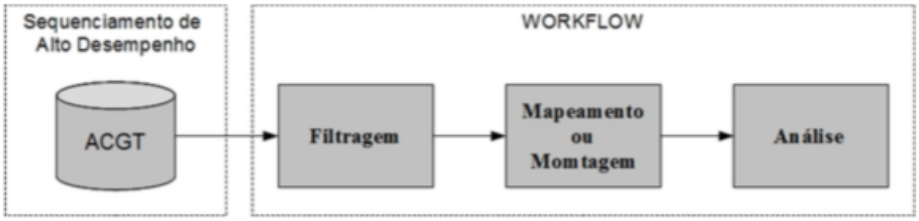

Figura 2.2: Exemplo de *workflow* para projetos genoma e transcritoma.

Estas *reads* geralmente tem um indicador de sua qualidade associado, o que pode ser usado para filtrar as *reads* com qualidade inferior, ou seja, *reads* ou partes delas, com certo grau de confiabilidade mínima. A etapa de filtragem pode também detectar contaminantes ou outros erros laboratoriais.

A filtragem, por se tratar de uma tarefa simples, pode ser feita tanto por *scripts* desenvolvidos localmente pelo usuário, com características simples de filtragem, como também por programas mais completos com diversas possibilidade de uso. Como exemplo pode-se citar o pacote *FASTX-Toolkit* (Fastx-Toolkit, 2015). Esse pacote fornece programas para tratamentos de arquivos *FASTA* e *FASTQ*, comumente utilizados para armazenar *reads*. Entre esses tratamentos encontra-se filtros e conversores de formato, úteis nessa fase.

Os arquivos da fase de filtragem podem somar *gigabytes* de dados, incluídos as *reads* e informações adicionais como o identificador para cada sequência e a qualidade de cada base mapeada. Esses dados constituem, em geral, parte considerável do espaço em disco ocupado em um experimento.

A fase de mapeamento tenta localizar, em um genoma de referência, as *reads* já processadas pela atividade de filtragem. O objetivo é encontrar o alinhamento para o maior número de *reads* possível e agrupá-las em porções maiores, que são analisados posteriormente. Essa técnica é usada para organismos que já possuem o seu DNA completamente (ou quase) sequenciado. Na Figura 2.3 é utilizado um genoma de referência conhecido para alinhar as *reads* obtidas descobrindo-se a sua posição sobre ele.

Figura 2.3: Exemplo do processo de mapeamento.

Quando não é conhecido o genoma de referência, pode-se alinhar as *reads* entre elas gerando sequências maiores chamadas *contigs*. Este processo é chamado de montagem. Na Figura 2.4 diversas *reads* são alinhadas e a sombra dessas *reads* mostra qual é porção do cromossomo sequenciado.

No caso de mapeamento de transcritos, a dificuldade ocorre a partir da existência de *splice junctions* nas *reads* sequenciadas. A Figura 2.5 mostra um exemplo do problema

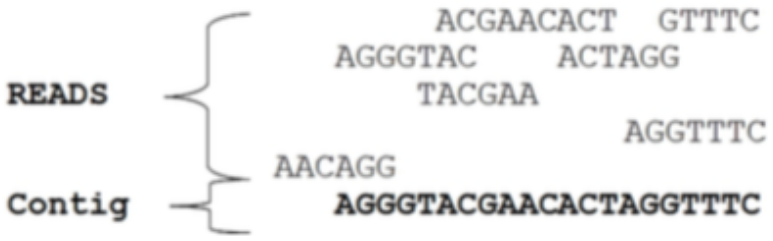

Figura 2.4: Exemplo do processo de montagem.

encontrado no processo de alinhamento de sequências de transcritos com *splice junctions*. Em (A) tem-se o processo de transcrição onde somente os *exons* são copiados para formar o transcrito. Em (B), quando a *read* com o transcrito é alinhada contra o genoma de referência, não encontra sua posição adequadamente porque existe um *intro* no genoma de referência. Quando isso ocorre, muitas *reads* podem ser descartadas por não encontrarem um alinhamento adequado.

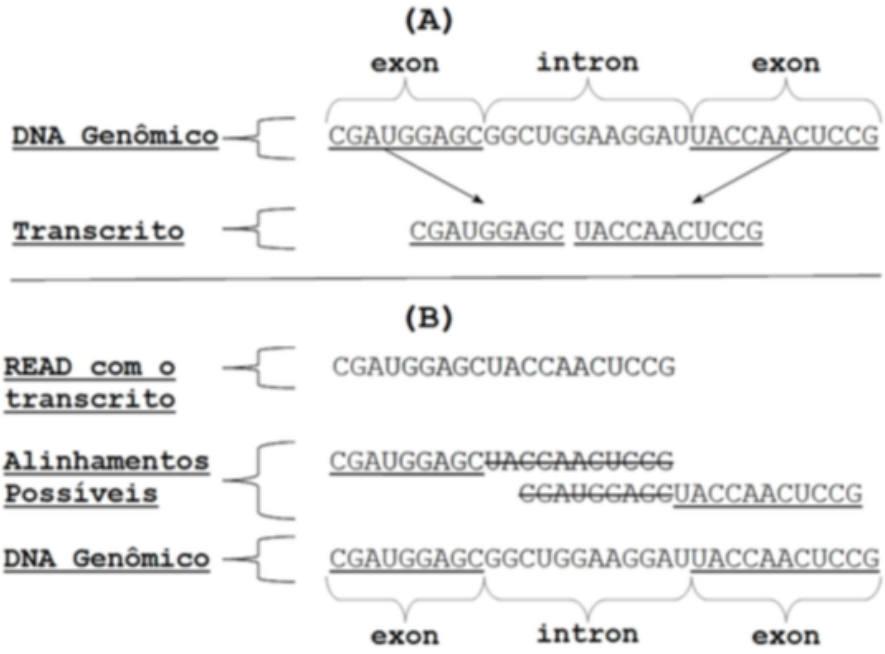

Figura 2.5: Exemplo do processo de transcrição (A) e do problema de alinhamento com *splice junctions* (B).

Como pode ser percebido, a fase de mapeamento requer algoritmos mais complexos que a filtragem. Assim, diversos programas podem ser encontrados na literatura, oferecendo diferentes recursos e especialidades. O programa Bowtie (Langmead et al., 2009), por exemplo, apresenta bons resultados com o tratamento de *reads* pequenas. Além disso ele foi desenvolvido para o tratamento eficiente de memória e foi projetado para utilizar mais de um processador durante o alinhamento, duas importantes características para o processo de alinhamento de grandes quantidades de *reads*. Outro exemplo é o TopHat

(Trapnell et al., 2009) que, apesar de utilizar o mesmo algoritmo do Bowtie, possui recursos específicos para o tratamento das *splice junctions*, tais como: identificar as uniões *exon-exon* e criar um banco de dados para ser utilizado durante o alinhamento. Outros programas também podem ser citados tais como CUFFLINKS (Roberts et al., 2011), especializado em alinhamento de sequências de RNA, e o Clustal (Higgins and Sharp, 1988), ferramenta utilizada para alinhamento múltiplo de sequências.

A última fase do sequenciamento é a fase de análise. Nesta fase tem-se grandes porções do DNA ou RNA mapeadas que podem ser analisadas por diferentes processos dependendo do objetivo do experimento. Como exemplo pode-se citar a busca pela diferença da expressão genética entre humanos suscetíveis e não suscetíveis a infecção de um determinado vírus. Neste caso faz-se o mapeamento de diversas amostras de RNA de indivíduos diferentes (suscetíveis e não suscetíveis) e na fase de anotação compara-se os transcritomas encontrados a fim de descobrir quais trechos expressos do DNA podem estar causando a deficiência dos indivíduos suscetíveis ou dando imunidade aos indivíduos não suscetíveis. A partir daí pode-se criar, por exemplo, formas de tratamento que ativem ou desativem a transcrição dessas porções de DNA.

Um exemplo de programa utilizado na fase de análise que pode ser citado é o R (R Development Core Team, 2009), que possui diversas ferramentas para análise estatística. Outro programa bastante utilizado neste fase é o BLAST (Altschul et al., 1990) utilizado para descobrir a existência de um gene em um determinado genoma e a sua funcionalidade.

2.3 Conceito de Proveniência de Dados

O termo proveniência de dados diz respeito à origem ou procedência dos dados. A proveniência também estar relacionada à auditoria, triagem, linhagem e origem do dado (Davidson and Freire, 2008). A proveniência ajuda a responder questões sobre os dados, tais como: quem criou este dado e quando, o momento em que o dado foi modificado, e por quem e qual foi o processo usado para criar o dado.

De acordo com Davidson and Freire (2008), a proveniência pode ser dividida em três tipos:

- Prospectiva: trata-se da sequência de processos utilizados (receita) para a geração do dado, ou seja, captura os passos que devem ser seguidos para a geração de um dado produto.
- Retrospectiva: trata-se das informações obtidas durante a execução dos processos de geração do dado. Compreende desde o tempo de duração de cada atividade executada até a origem dos dados de entrada. Além disso, não depende do tratamento da proveniência prospectiva para ser utilizado. Em outras palavras, é como se fosse um *log* detalhado da execução de uma tarefa.
- Dados definidos pelo usuário: qualquer informação que o usuário julgar necessária para futura utilização. Como exemplo, pode-se citar anotações, conclusões a respeito do processo e, até mesmo, observações sobre parâmetros utilizados.

Segundo Tan (2004), a obtenção da proveniência pode seguir duas abordagens, abordagem preguiçosa (*lazy*) na qual a obtenção da proveniência é executada somente no

momento que é solicitada, e abordagem ansiosa (*eager*) na qual a proveniência é obtida durante a geração da informação e é armazenada para permitir futuras consultas. Cabe ressaltar que as duas abordagens podem ser combinadas, fazendo com que algumas informações sejam obtidas pela abordagem preguiçosa e outras pela ansiosa.

Ainda segundo Davidson and Freire (2008), quando a proveniência é capturada de forma automática, pode-se dividí-la nos níveis:

- *Workflow*: envolve a descrição da execução de um processo, ou seja, das tarefas que dele fazem parte, é usado pela grande maioria das soluções com SGWfC (Sistemas de Gerência de *Workfow* Científicos) e nesse caso deve ser adaptado para capturar os dados dos diferentes processos executados;
- Atividade: pode ocorrer de duas formas. Na primeira, cada processo/programa executado é alterado para capturar os dados de proveniência. Na segunda, podem ser criados programas específicos para monitorar a execução de um determinado processo e capturar os dados de proveniência;
- Sistema Operacional: utiliza os dados fornecidos pelo próprio sistema operacional como insumo para a proveniência.

2.3.1 Modelos de Proveniência de Dados

Modelos de proveniência de dados têm como principal objetivo fornecer uma estrutura para que os dados de proveniência possam ser armazenados e recuperados, mantendo seu significado e potencializando os seus benefícios. Com o objetivo de definir os requisitos necessário para aplicação do modelo em projetos de bioinformática, considerou-se as seguintes características (de Paula et al., 2013):

- Objetivo do modelo de proveniência;
- Representação gráfica simplicada para que o usuário possa rever seus experimentos de forma prática;
- Modelo de proveniênica bem definido;
- Fornecimento de ferramentas que facilitem a utilização do modelo de proveniência;
- Capacidade do modelo de proveniência de propiciar o intercâmbio de informações entre diferentes sistemas;
- Capacidade de difusão do modelo de proveniência.

Diversos modelos de dados são encontrados hoje na literatura, com diferentes formatos e objetivos. A seguir é feita uma breve revisão sobre os modelos de proveniência de dados a fim de verificar os pontos relevantes de cada um. Os modelos apresentados foram escolhidos devido a sua relevância para a aplicação de projetos de bioinformática e por serem os mais utilizados na literatura.

- *W7* (Ram and Liu, 2007): tem como base a ontologia de Bunge (Bunge, 1977), a qual objetiva descrever as propriedades de um objeto de caráter geral. A partir deste

estudo, o modelo *W7* estruturou a proveniênica de uma peça de dado através da resposta a 7 perguntas (ou dimensões): O que?, Quem?, Quando?, Onde?, Como?, Qual?, e Por quê?.

- *Provenance Vocabulary* (Hartig and Zhao, 2010): volta sua atenção para o problema da proveniência de dados publicados na web. A sua principal característica é fornecer classes e propriedades para que publicadores de dados para *web* possam armazenar, além dos dados publicados, também os metadados com informações úteis sobre a proveniência dos dados publicados.

- *Provenir Ontology* (Sahoo and Sheth, 2009): foi desenvolvido para ser um modelo de proveniência de dados genético, priorizando a interoperabilidade entre diferentes sistemas e sua adaptação para qualquer aplicação. Da mesma forma que no modelo *Provenance Vocabulary*, o modelo *Provenir Ontology* define um núcleo comum e permite a criação de módulos específicos para o domínio da aplicação desejada.

- OPM (*Open Provenance Model*): começou a ser discutido em maio de 2006 no *Workshop* Internacional de Anotação e Proveniência. É um modelo aberto voltado a caracterização da proveniência de qualquer "coisa", material ou imaterial. O modelo OPM, descrito em (Moreau et al., 2009) procura demonstrar a relação causal entre eventos que afetam objetos (digitais ou não) e descreve essa relação através de um grafo acíclico direcionado.

- PROV-DM (*Provenance Data Model*): tem como principal função descrever as pessoas, entidades e atividades envolvidas na produção de um dado. Além disso, o modelo cria as condições para que a proveniência seja demonstrada e trocada entre diferentes sistemas. Apesar do modelo PROV-DM utilizar os mesmos princípios do OPM, largamente aplicado em projetos apresentados na literatura, ele possui um maior detalhamento permitindo demonstrar a proveniência de forma mais precisa.

O modelo PROV-DM se destaca entre os demais devido a possuir as características necessárias para a aplicação em projetos de bioinformática, tais como uma representação gráfica adequada e capacidade de proporcionar o intercâmbio de informações entre diferentes sistemas. Além de se tornar uma recomendação da *W3C* (*World Wide Web Consortium)* sendo a principal organização de padronização da *World Wide Web*.

2.3.2 PROV-DM - *Provenance Data Model*

O PROV-DM teve a sua primeira versão desenvolvida em outubro de 2011 sendo uma recomendação do W3C. Até o momento da escrita deste trabalho, a versão mais nova do PROV-DM foi publicada em abril de 2013. Tem como principal característica demonstrar a proveniência de qualquer objeto (real ou imaginário) através de um grafo direcionado. A raiz deste grafo representa a entidade cuja proveniência está sendo representada e as arestas são direcionadas para as atividades e entidades das quais foram originadas.

O modelo é dividido em oito elementos que contém tanto os elementos como as relações possíveis entre eles (W3C, 2015):

- Entidades (*Entities*): representa qualquer objeto (real ou imaginário);

- Atividades (*Activities*): representa os possíveis processos executados que deram origem ao objeto foco da proveniência.
- Agente (*Agents*) e responsabilidades: são entidades que influenciam, direta ou indiretamente, a execução das atividades, recebem atribuições de outros agentes e podem ter algum tipo de ligação (posse, direitos, etc...) sobre outras entidades;
- Derivações (*Derivations*): descreve a relação entre diferentes entidades durante o ciclo de transformação executado pelas atividades permitindo demonstrar a dependência entre as entidades usadas e geradas;
- Coleções (*Collection*): representa um conjunto de Entidades, e pode ter a sua proveniência representada como um conjunto, independente da proveniência do seu conteúdo;
- Anotações (*Annotation*): fornece mecanismos para inclusão de anotações para os elementos do modelo;
- Plano (*Plan*): representa um conjunto de ações ou passos que um Agente deve seguir para chegar a um determinado objetivo;
- Conta (*Account*): representa um conjunto de informações (tipos e relações) que compõe um grafo de proveniência.

A proveniência pode ser definida como um grafo acíclico dirigido (DAG). Os nós no grafo podem representar objetos, tais como arquivos, programas e pessoas. Esses nós podem possuir atributos. Por exemplo, um nó que representa um programa pode ser anotado com atributos como a data de execução, a linha de comando e a versão do programa. Já as arestas entre os nós indicam que há uma dependência entre os objetos. Por exemplo, uma aresta de um objeto A para um objeto B pode indicar que B foi derivado de A. O grafo de proveniência, por definição, é acíclico, ou seja, não há presença de ciclos entre os objetos.

A Figura 2.6 ilustra os símbolos utilizados pelo modelo PROV-DM para representar os diferentes nós do grafo. O símbolo da Entidade também é utilizado para representar o tipos Coleção, uma vez que representa subtipo do tipo Entidade. O tipo Conta não tem símbolo pois representa o próprio grafo de proveniência.

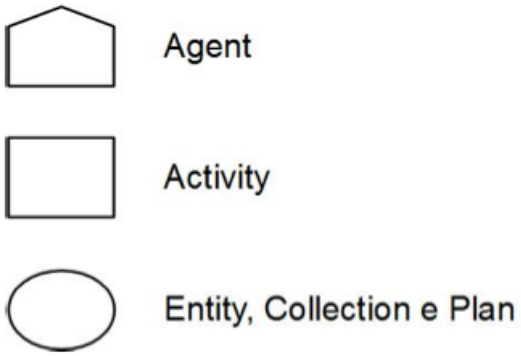

Figura 2.6: Representação gráfica dos nós do modelo PROV-DM (W3C, 2015).

As relações da Figura 2.7 representam as arestas no grafo de proveniência que, por sua vez, indicam as relações possíveis entre cada nó. Além de descrever cada tipo que compõe o modelo, os seis componentes também detalham as relações que podem ocorrer entre cada um dos tipos. O modelo PROV-DM possui mais relações/arestas, porém serão descritas aqui apenas as mais relevantes para este trabalho:

- *wasDerivedFrom*: indica, de forma geral, que uma Entidade (original) foi usada, direta ou indiretamente, na geração de outra Entidade (derivada);
- *used*: indica que uma Entidade foi usada por uma Atividade;
- *wasGeneratedBy*: indica que uma Entidade foi gerada por uma Atividade;
- *wasAssociatedWith*: atribui algum tipo de responsabilidade a um Agente sobre uma Atividade;
- *memberof*: Indica que uma determinada Entidade é membro de uma Coleção.

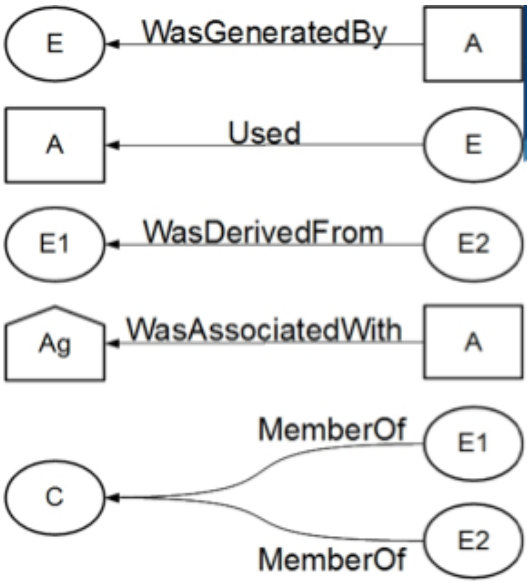

Figura 2.7: Representação gráfica das relações do modelo PROV-DM (W3C, 2015).

A fim de permitir um histórico mais detalhado das derivações feitas durante o processo representado pela proveniência, a relação *wasDerivedFrom* possui diversos subtipos. A seguir tem-se a descrição dos subtipos da relação *wasDerivedFrom* (W3C, 2015):

- *wasQuotedFrom*: indica que a Entidade derivada foi gerada a partir da cópia de parte da Entidade original;
- *wasRevisionOf*: indica que a Entidade derivada foi gerada a partir da revisão da Entidade derivada, sendo que tal responsabilidade da revisão pode ser atribuída a um Agente;
- *tracedTo*: indica, de forma genérica, que existe uma relação de dependência entre duas Entidade, sem especificar qual é a entidade origem e a entidade destino.

As restrições do modelo PROV-DM descrevem restrições para a construção de grafos de proveniência. Tais restrições tem dois objetivos principais: (a) evitar a formação de grafos inválidos ou inconsistentes e (b) fazer inferências sobre os elementos e as relações entre eles.

As restrições são divididas em três grupos: restrições de Atividade, restrições de Entidade e restrições de Agente. A seguir são descritas as restrições relevantes para este trabalho:

- Restrições de Atividade: restrições relacionadas à execução das Atividades, tais como:
  - Início/Fim: o início da execução de uma Atividade deve preceder o seu fim;
  - Uso: o uso de uma Entidade por uma Atividade deve ocorrer entre o início e o fim da sua execução;
  - Geração: a geração de uma Entidade por uma Atividade deve ocorrer entre o início e o fim da sua execução.
- Restrições de Entidade: restrições relacionadas ao ciclo de vida de uma Entidade, exemplos:
  - Geração/Uso: a geração de uma Entidade deve preceder o seu uso;
  - Derivação/Uso/Geração: para os casos em que existe uma derivação entre duas Entidades, por exemplo E2 é derivado de E1, e o uso de E1 é conhecido, então o uso de E1 deve preceder a geração de E2;
  - Derivação/Geração/Geração: para os casos em que existe uma derivação entre duas Entidades, por exemplo E2 é derivado de E1, e o uso de E1 não é conhecido, então a geracção de E1 deve preceder a geração de E2.
- Restrições de Agente: restrição relacionada ao ciclo de vida de um Agente.
  - Associação: a associação entre um Agente e uma Atividade deve ocorrer entre o início e o fim da execução desta Atividade.

O modelo PROV-DM fornece um recurso para adição de informações extras no grafo de proveniência através de um identificador chamado Nota (*Note*). Este identificador representa um conjunto de pares atributo-valor que permite ao usuário criar diversos tipos de anotações. Cada Nota, por sua vez, pode ser conectada a qualquer tipo ou relação existente no grafo a partir de uma relação chamada *hasAnnotaion*. A Figura 2.8 mostra um exemplo de grafo baseado no modelo PROV-DM, no qual podem ser vistas quatro anotações, sendo duas relacionadas às arestas *wasAssocitedWith* informando o papel de cada Agente na Atividade, uma relacionada com a Atividade e outra relacionada com a Entidade gerada.

2.3.3 PROV-DM em *Workflow* de Bioinformática

Projetos de bioinformática tem como características: grande volume de dados processados, a execução de diversos processos com diferentes opções de programas e parâmetros

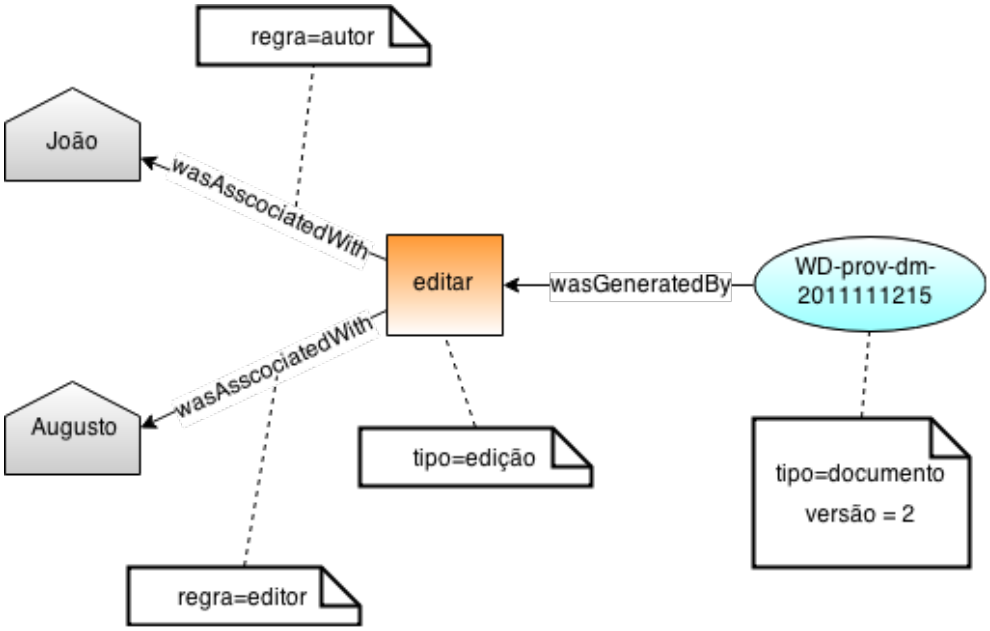

Figura 2.8: Exemplo de grafo do modelo PROV-DM adaptado do (W3C, 2015).

de configuração e um fluxo de dados entre diversos arquivo com diferentes formatos. Tais características afetam de forma significativa a análise dos resultados. Portanto, manter a proveniência de dados em projetos de bioinformática requer uma solução que permita armazenar a ligação entre os dados processados juntamente com as informações das execuções de cada processo e de seus resultados. Além disso, em laboratórios onde diversos usuários podem estar trabalhando em diferentes projetos, torna-se importante o registro de quem executou o quê. Logo, os dados, os processos executados e as pessoas envolvidas formam os três pilares do modelo PROV-DM.

O PROV-DM já foi aplicado no contexto da Bioinformática em (Paulino et al., 2010), (de Oliveira et al., 2012), (Costa et al., 2013), (Gonçalves et al., 2013), e (Oliveira et al., 2014). Neste trabalho, a escolha do PROV-DM foi baseada em de Paula et al. (2013) onde é apresentada uma comparação entre os principais modelos de proveniência de dados disponíveis na literatura. Os principais requisitos considerados foram:

- Capacidade de representar a proveniência de uma peça de dado, descrevendo os processos e insumos utilizados em sua geração;
- Uma representação gráfica adequada, com diferentes símbolos para cada elemento, e relações suficientes para demonstrar a proveniência de forma objetiva;
- Símbolo para representar grandes conjuntos de dados;
- Capacidade do modelo de proporcionar o intercâmbio de informações entre diferentes sistemas.

Dessa forma, de acordo com de Paula et al. (2013) a aplicação do modelo PROV-DM para representar a proveniência de dados em projetos de bioinformática, se mostrou bastante simples e direta. Os componentes do modelo, tais como o agente, atividade e coleção, representam elementos presentes em grande parte dos experimentos executados

em projetos de bioinformática. As relações, por sua vez, demonstram de forma objetiva as dependências entre cada elemento no grafo, e a utilização das regras e do tipo de derivação permitem maior grau de especificidade quando necessário.

Porém, o modelo PROV-DM prevê somente a estrutura básica dos nós e relações para representar a proveniência de dados. Para uma definição completa se faz necessário a inclusão de dados capazes de caracterizar cada elemento do modelo, além da inclusão de uma entidade capaz de manter agrupados diferentes experimentos que estejam relacionados entre si.

Uma vez que os projetos de bioinformática podem possuir diversos experimentos foi definido em de Paula et al. (2013) um novo elemento, não previsto no modelo PROV-DM, chamado *Project*. A criação deste elemento vem da necessidade de agrupar estas diferentes execuções de experimentos relacionados ente si permitindo que o usuário crie agrupamento de execuções do experimento.

Logo, além da representação gráfica do grafo de proveniência, algumas informações adicionais são necessárias para entender completamente um determinado experimento. Assim, os elementos projeto e conta foram incluídos no modelo para representar, respectivamente, um projeto de bioinformática e a execução de um experimento. Também foi incluído um conjunto mínimo de informações relacionadas a cada um dos elementos conforme detalhado na 2.1 (de Paula et al., 2013).

2.4 Modelo de Dados de Grafo

Um grafo $G(V, E)$ consiste em um conjunto finito não vazio de vértices V, e um conjunto $E \subseteq \{VxV\}$ de arestas. Ou seja, dado $V = \{v_1, v_2, v_3, ..., v_n\}$, um conjunto de relacionamentos possíveis seria $E = \{(v_1, v_2), (v_1, v_3), (v_2, v_6), ..., (v_n - 1, v_n)\}$ no qual cada subconjunto seria uma aresta (West et al., 2001).

Grafos permitem uma representação gráfica, onde os vértices são pontos e arestas linhas ligando-os, conforme a Figura 2.9. Neste exemplo, os vértices representam pessoas ligadas por arestas que representam o relacionamento de "seguir" entre elas. Além disso, o exemplo ilustra como os vértices podem ser apresentados com rótulos para dar um melhor significado, assim como as arestas, indicando um tipo de relação.

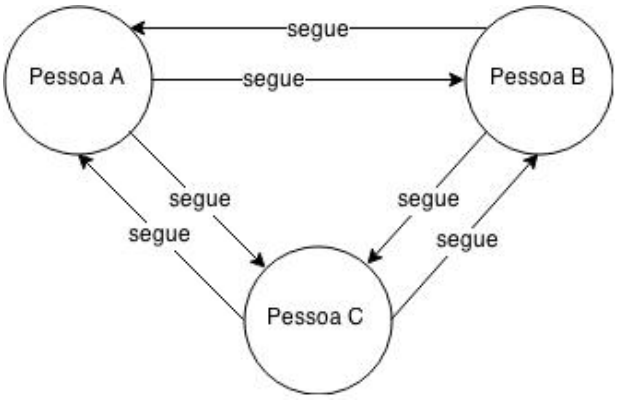

Figura 2.9: Exemplo de grafo.

Tabela 2.1: Informações do grafo de proveniência para projetos de bioinformática.

Projeto		Experimento	
Nome	Nome do projeto	Nome	Nome do experimento
Descrição	Descrição do projeto	Descrição	Descrição do experimento
Instituições Financiadoras	Lista das instituições que financiaram o projeto	Local	Local de execução
Instituições participantes	Lista das instituições que participam do projeto	Data Inicial	Data inicial da execução
Coordenador	Nome do coordenador do projeto	Data Final	Data final da execução
Data Inicial	Data de início do projeto	Versão e Data	Número e data da versão gravada
Data Final	Data final do projeto	Anotações	Qualquer informação adicional sobre o projeto
Agente		**Atividade**	
Nome	Nome do usuário	Nome	Nome da atividade
Instituição	Instituição do usuário	Programa	Nome do programa
Cargo	Cargo ou função	Versão	Versão do programa
Função	Função no experimento	Comando	Linha de comando com os parâmetros utilizados
Grupos	Grupos para filtrar o grafo de proveniência	Função	Descrição do que a atividade executou
Anotações	Qualquer informação adicional sobre o agente	Hora Inicial	Data e hora em qua a atividade foi iniciada
		Hora Final	Data e hora em que a atividade foi concluída
		Ambiente	Descrição do ambiente computacional em que a atividade executou
		Grupos	Grupos para filtrar o grafo de proveniência
		Anotações	Qualquer informação adicional sobre a entidade

O modelo de dados de grafo é uma estrutura na qual o esquema e/ou instâncias são modeladas como um grafo dirigido, possivelmente rotulado, ou generalizações da estrutura de dados do grafo, onde a manipulação de dados é expressa por operações orientadadas para o grafo e construtores de tipos, e restrições de integridade que podem ser definidas

sobre a estrutura do grafo (Angles and Gutierrez, 2008).

Normalmente, os bancos de dados de grafos são aplicados onde a informação sobre a interconectividade entre os dados ou topologia é mais importante, ou tão importante, quanto os dados. Exemplos de aplicações onde bancos de dados de grafos podem ser aplicados é a interação genética, onde nós representam proteínas, e as arestas representam as interações físicas entre proteínas. Aplicações que envolvem a análise de redes para sistemas de recomendação, com o objetivo de apresentar informações que possar ser interessantes para os usuários com base em conhecimento prévio extraído do ambiente social. E aplicações que tem colaboração interativa e compartilhamento de informações, tais como *Facebook*, *Linkedln* e *Flickr angels*.

As principais vantagens de se usar esse tipo de modelo, segundo Angles and Gutierrez (2008) são:

- Permite uma modelagem mais natural dos dados porque as estruturas de grafos são intuitivas para o usuário;
- Permite expressar as consultas em um nível maior de abstração;
- Permite a implementação de algoritmos eficientes para realizar específicas operações.

Quanto aos componentes de um modelo de dados de grafo pode-se ter (Dominguez-Sal et al., 2011):

- Atributos: são informações associadas a esses nós e/ou arestas, podendo ser uma *string* ou valores numéricos, que indicam as características da entidade ou relação. Para o caso particular de arestas, alguns grafos incluem atributos numéricos que quantificam a relação, na qual é geralmente interpretada como o comprimento, peso, custo ou a intensidade da relação. Essa característica permite que certas informações sejam armazenadas próximas aos nós, facilitando a recuperação do dado. Por exemplo, na Figura 2.10 os nós possuem atributos como nome do tipo *String*, tempo inicial do tipo *Date* e tamanho do arquivo do tipo *Long*;
- Direcionamento: dependendo do problema o relacionamento entre dois nós pode ser simétrico ou não. Se o relacionamento é simétrico, as duas pontas da aresta são diferentes, mas indistinguíveis, ou seja, não há ponta inicial nem ponta final. Caso o relacionamento não seja simétrico, é possível diferenciar as duas pontas da aresta, em outras palavras, a ponta inicial da aresta é o nó a partir do qual começa a aresta, e a ponta final da aresta é o nó na qual a aresta termina. Por exemplo, na Figura 2.10 o relacionamento Usou não é simétrico já que é fácil distinguir o nó inicial e o nó final;
- Nós e arestas rotuladas: em algumas aplicações, é possível diferenciar rótulos (ou tipos) de nós e arestas. Rotulagem tem um impacto importante porque alguns aplicativos requerem distinção entre os diferentes tipos de relacionamento, pois permite que alguns bancos de dados em grafos utilizem o rótulo para filtrar os vértices em consultas. Por exemplo, os nós da Figura 2.10 são rotulados pela propriedade tipo que os diferencia. Já as arestas são rotuladas com os *labels* FoiAssociadoCom e Usou.

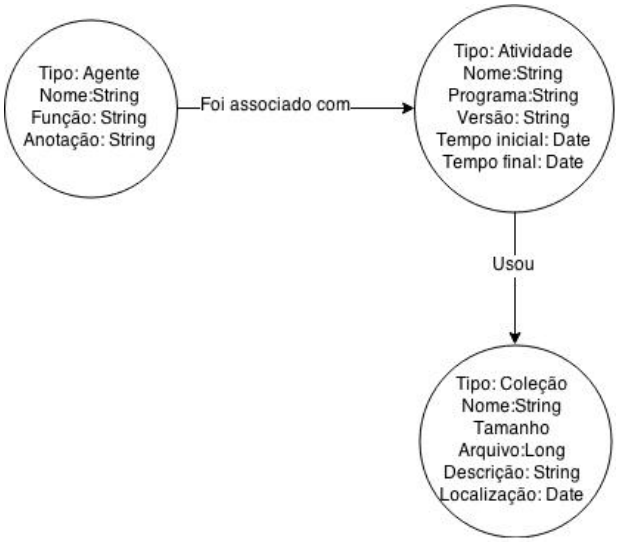

Figura 2.10: Exemplo de grafo de atributo.

Conforme Dominguez-Sal et al. (2011) e Angles (2012a) os bancos de grafos podem ser classificados em:

- Grafo de Atributos: permite que tanto os vértices como as arestas armazenem informações de atributos como um par de chave e valor. Essa característica permite que certas informações sejam armazenadas próximas aos nós, facilitando a recuperação do dado, além de simplificar a diagramação do modelo, pois são necessários menos vértices, caso alguns atributos não sejam utilizados para expressar relacionamentos entre as instâncias. Uma vantagem é beneficiar certos tipos de consultas que realizam buscas sobre atributos. Um exemplo de grafo de atributos é a Figura 2.10. Percebe-se que o uso de atributos deixa o diagrama mais próximo ao de um modelo relacional. Essa semelhança pode causar problemas ao se manter, pois ao se manter todos os atributos juntos aos nós ou arestas, perde-se o benefício de enfatizar os relacionamentos.
- Grafo Simples: o modelo de dados tem apenas vértices e arestas cabendo aos rótulos armazenar os valores identificadores das instâncias. Dessa forma os atributos do grafo transformam-se também em nós, como se fossem entidades. O que cria um modelo mais complexo com uma quantidade maior de relacionamento, porém mais normalizado, pois qualquer atributo que possa ser duplicado é compartilhado, mantendo-se apenas uma instância. A desvantagem dessa modelagem é o aumento da quantidade de elementos no modelo. Além disso algumas consultas também podem ficar mais complexas pois deve-se buscar os vértices ligados a uma instância se for necessário verificar algum atributo. A Figura 2.11 apresenta um modelo de um grafo simples. Pode-se ver que os atributos da entidade Agente tornaram-se nós.
- Multigrafos: diferem dos grafos em que dois nós podem ser conectados por várias arestas. Por exemplo, considerando as cidades de um mapa como nós e as estradas entre elas como arestas, dois nós poderão ter múltiplas conexões, conforme mostra a Figura 2.12, onde há duas aresteas (estradas) entre os nós Fortaleza e Brasília.

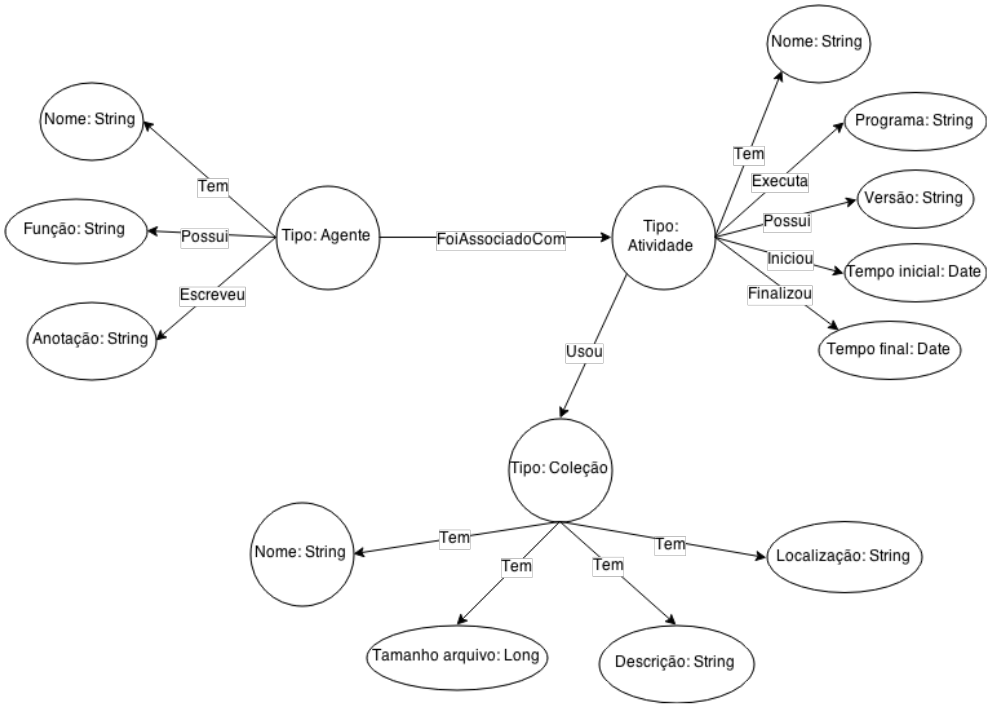

Figura 2.11: Exemplo de grafo simples.

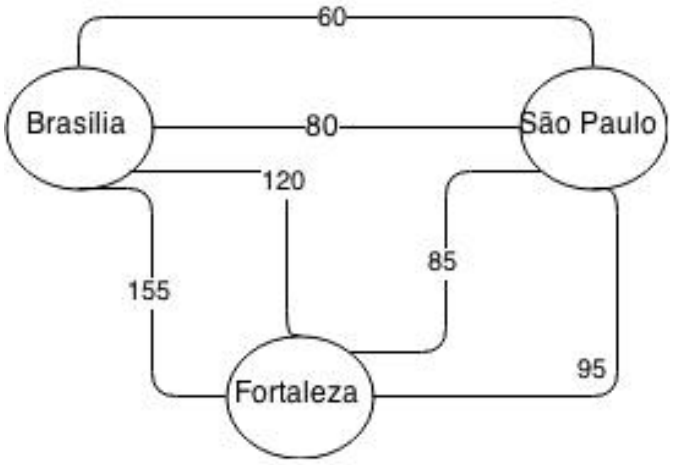

Figura 2.12: Exemplo de multigrafo.

- Hipergrafos: um hipergrafo permite que suas arestas liguem mais de dois vértices. Ou seja, a relação entre seus vértices não precisa ser binária. Isso permite que em certas situações onde o mesmo relacionamento ocorre entre várias instâncias seja utilizando apenas um arco. Pode ser usado para modelagem de problemas relacionado ao fluxo de atividades mostrando como representar um fluxo de tarefas. Por exemplo, com base na Figura 2.13 pode-se descrever como ocorre a execução dos processos sobre o grafo. A execução inicia-se pelo passo P1. Quando a execução chega em P2, uma decisão deverá ser tomada. Executam-se, então, os passos 3 ou 4 ou os passos 6 e 7. Se a condição de P2 for satisfeita, então executam-se os passos 3 e 4, caso contrário os passos 6 e 7 serão executados. Logo, P2 gera um ou-exclusivo,

pois se P3 e P4 forem executados, então P6 e P7 não serão e vice-versa.

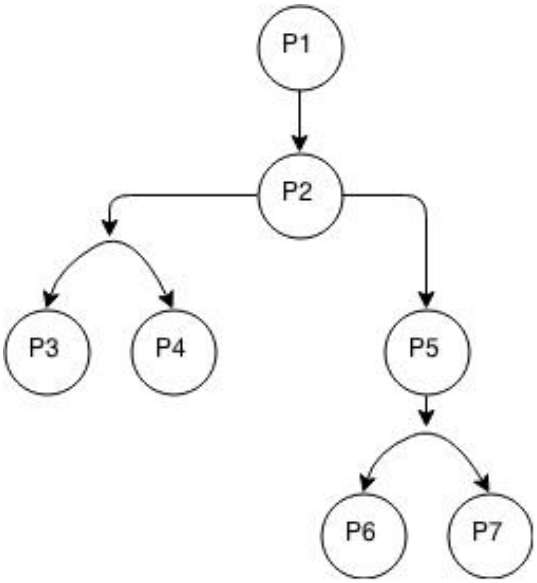

Figura 2.13: Exemplo de hipergrafo.

- Grafo aninhado: são grafos que suportam outros grafos como nós. Um grafo pode ser vinculado a outro, como se fosse um vértice. Um grafo agrupado torna-se um hipernó, sendo tratado como um vértice comum, e o grafo passa a ser chamado de aninhado (Angles and Gutierrez, 2008). Na Figura 2.14, tem-se um grafo de proveniência de dados resultado da execução do experimento que trata da expressão diferencial entre células de rim e fígado. Fica claro, que os nós e arestas referentes às atividades executadas para tratamento diferencial do rim pertencem a um nó maior que agrupa as etapas relacionadas, assim como o nó Grupo Fígado representa o agrupamento das atividades executadas na etapa referente a expressão diferencial do fígado.

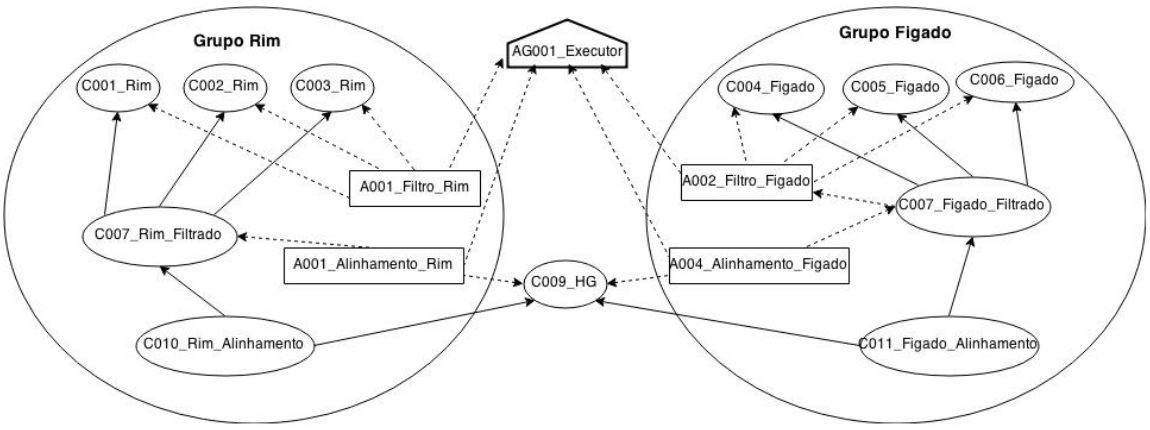

Figura 2.14: Exemplo de grafo aninhado.

Ainda, de acordo com Dominguez-Sal et al. (2011), há vários tipos de operações em grafos. Essas operações não são típicas de um domínio específico, mas são operações que podem ser necessárias em algum contexto. As principais operações são:

- Travessias: são operações que se iniciam de um único nó e explora recursivamente os vizinhos até que uma condição final, tais como a profundidade ou visitar um nó de destino seja alcançado. Por exemplo, considerando a operação de calcular o menor caminho, que é a menor sequência de arestas que conecta dois nós. Essa operação pode ser usada para sistemas de recomendação ou *ranking*;
- Encontrar componentes: um componente conectado é um subconjunto de nós do grafo onde existe um caminho entre qualquer par de nós. Encontrar componentes conectados geralmente é crucial em muitas operações, normalmente utilizados em uma fase de pré-processamento. Além disso, algumas operações são úteis para estudar a vulnerabilidade de um grafo, ou a probabilidade de separar um componente conectado em dois outros componentes;
- A correspondência de padrões: reconhecimento de padrões lida com algoritmos que visam reconhecer ou descrever padrões de entrada. Padrões de grafos são geralmente classificados em exato ou aproximado;
- Equivalência estrutural: refere-se à medida em que nós têm um conjunto de ligações com outros nós do sistema. Encontrar semelhança entre nós em um grafo tem se mostrado muito importante em problemas de análise de redes socias;
- Transformação: compreende as operações que alteram o banco de dados de grafo, tais como cargas de um grafo, adicionar/remover nós ou arestas dos grafos, criar novos tipos de nós/arestas/ atributos ou modificar o valor de um atributo;
- Acesso em cascata: uma operação segue um padrão em cascata se a consulta executa operações com uma profundidade de pelo menos dois nós. Já uma operação que não seja em cascata pode acessar apenas um nó, uma aresta ou vizinhos de um nó;
- Escala: classificam-se as consultas, dependendo do número de nós acessados. Pode-se distinguir dois tipos de consultas, consultas globais e de proximidade. Em outras palavras, considera-se como consultas globais aquelas que acessam todos os nós ou as arestas do grafo, já as de proximidade acessam apenas uma parte do gráfico;
- Resultado: diferenciam-se três tipos de resultados, grafos, agregados e conjuntos. A saída mais comum para uma consulta de banco de dados grafo é outro grafo, que é normalmente uma transformação, uma seleção ou uma projeção do grafo original, que inclui nós e arestas. O segundo tipo de resultados é a construção de agregados, cuja aplicação mais comum é resumir as propriedades do grafo. Finalmente, um conjunto é uma saída que contém tanto as entidades atômicas ou conjuntos de resultados que não estão organizados na forma de grafos.

2.5 Banco de dados de Grafo em *NoSQL*

Os bancos de dados baseados em grafos não são tecnologias recentes, vários trabalhos foram realizados ao longo das últimas décadas, embora muito pouco tenha sido produzido

durante a primeira década do século XXI. Logo após esse período, com a necessidade de manipular dados de natureza típica de grafos e os bancos *NoSQL*, ocorreu uma revitalização nessa área, ampliando o interesse e aplicação desses sistema.

Os bancos de dados *NoSQL* trouxeram abordagens diferenciadas do modelo relacional para organizar os dados. Atualmente, esses bancos se dividem em quatro grupos baseados na estratégia de armazenamento (Padhy et al., 2011):

- Chave/Valor: os dados são armazenados como pares chave-valor que são indexados para recuperação por chaves. Os mais populares são o *Riak* (Riak, 2014), *Redis* (Redis, 2015), *Berkeley DB* (Olson et al., 1999), *Amazon DynamoDB* (Dynamodb, 2015), *Porject Voldemort* (Voldemort, 2015) e *HamsterDB* (HamsterDB, 2015).
- Orientado a coluna: armazena os dados em linhas com colunas associadas, fazendo uso de uma chave de linha. Famílias de colunas são grupos de dados relacionados que, frequentemente, são acessados juntos (Sadalage and Fowler, 2013). Exemplos são o *BigTable* (Chang et al., 2008), *Cassandra* (Cassandra, 2015) e *HBase* (HBase, 2015).
- Armazenamento baseado em documentos: os dados são armazenados e organizados como uma coleção de documentos. Eles armazenam documentos baseados no formato *JSON*, XML, BSON, entre outros. Exemplos são o *MongoDB* (MongoDB, 2015), *Apache CouchDB* (CouchDB., 2015) e *RavenDB* (RavenDB, 2015).
- Bancos de dados de grafos: atividades sobre bancos de dados de grafos surgiram na primeira metade dos anos noventa, porém entraram em desuso. Resurgindo recentemente devido a projetos onde este estilo de base de dados é necessário. Por exemplo química, biologia, mineração, redes sociais e semântica web(Angles, 2012b). Eles permitem armazenar entidades (nós) e também relacionamentos entre essas entidades (arestas), facilitando consultas típicas dessa estrutura. Exemplos são *Neo4J* (Neo4J, 2015), *Infinite Graph* (InfiniteGraph, 2015) ou *FlockDB* (FlockDB, 2015).

Ao se trazer os grafos para os sistemas de bancos *NoSQL*, cada iniciativa construiu sua implementação da forma que considerou mais adequada, porém, ainda que nos detalhes eles possam variar, certas características se mantiveram semelhantes. O *Neo4J* e o *OrientDB*, por exemplo, possuem o conceito de vértices e arestas com atributos, embora o *OrientDB* também implemente o modelo de dados dos *NoSQL* de documentos.

2.6 Banco de Dados Relacional *Versus* Grafo

Esta seção apresenta uma análise comparativa entre o banco de dados baseado em grafo e o banco de dados relacional do ponto de vista da eficiência e do armazenamento de dados de proveniência em *workflow* de bioinformatica.

Um estudo feito por Vicknair et al. (2010) compara uma base de dados relacional, como MySQL, com o modelo de grafos, tal como *Neo4J*, para armazenamento de dados de proveniência. Ele propõe um *benchmark* objetivo e uma comparação subjetiva baseada na documentação e experiência. Os testes objetivos incluem a velocidade de processamento em um conjunto de consultas pré-definidas, requisitos de espaço em disco e escabilidade.

Já, os testes subjetivos incluem maturidade/nível de suporte, facilidade de programação, flexibilidade e segurança.

A comparação objetiva mostrou que os bancos de dados de grafos apresentaram melhor desempenho do que os bancos de dados relacionais em consultas estruturais e que envolvem busca de textos. Entretanto, devido ao fato do mecanismo de indexação usado em bancos de grafos ser baseado em *strings*, as consultas que envolvem contagem numérica apresentaram menor eficiência.

Já a comparação subjetiva mostrou que em relação ao nível de maturidade/suporte os bancos de dados de grafos, em geral, possuem um mercado menor, consequentemente menos usuários, ausência de uma linguagem unificada e suporte. Em relação à facilidade de programação, depende do problema, para consultas transversais no grafo é mais simples em bancos de dados de grafos enquanto procurar valores de atributos em uma tabela é extremamente fácil com o modelo relacional.

Quanto a flexibilidade o modelo de grafos, especificamente o *Neo4J*, tem um esqema facilmente mutável. O esquema do modelo relacional pode ser alterado, porém fazer isso é mais complexo do que no modelo de grafos. E por fim, quanto a segurança, o modelo relacional possui suporte interno a multi-usuários, por outro lado, muitas bases de dados de grafos não possuem suporte para ambientes multi-usuários. Portanto, de acordo com Vicknair et al. (2010), o modelo de dados de grafos ainda se apresenta prematuro para ser executado em um ambiente de produção.

No contexto de *workflow* da bioinformática foi apresentado em Pinheiro et al. (2013) o armazenamento de dados de proveniência em banco de dados relacionais. Este modelo será a base de comparação com o modelo de dados proposto nesta dissertação que é baseado em grafo. De acordo com a Figura 2.15 tem-se as tabelas:

- *Collection_ Provenance*: representa uma coleção de entidades, ou um arquivo usado em uma atividade. Um experimento pode envolver diversos arquivos que são usados na execução de uma atividade;
- *Was_ Derived_ From*: representa as relações entre arquivos, designando quando um arquivo é derivado de outro;
- *Used*: representa a relação na qual uma atividade usou um arquivo;
- *Was_ Generated_ By*: representa que um arquivo foi gerado a partir de outro arquivo;
- *Was_ Controlled_ By*: representa a relação que uma atividade foi executada/controlada por um agente;
- *Entity_ Provenance*: representa o dado unitário, por exemplo uma sequência de DNA ou alinhamento.
- *Experiment*: representa a execução do experimento;
- *Institution*: modela as instituições participantes dos experimentos;
- *Agent*: representa o responsável por uma atividade;
- *Experiment_ Agent*: representa os experimentos de um agente;
- *Activity*: representa uma etapa ou atividade do *workflow*.

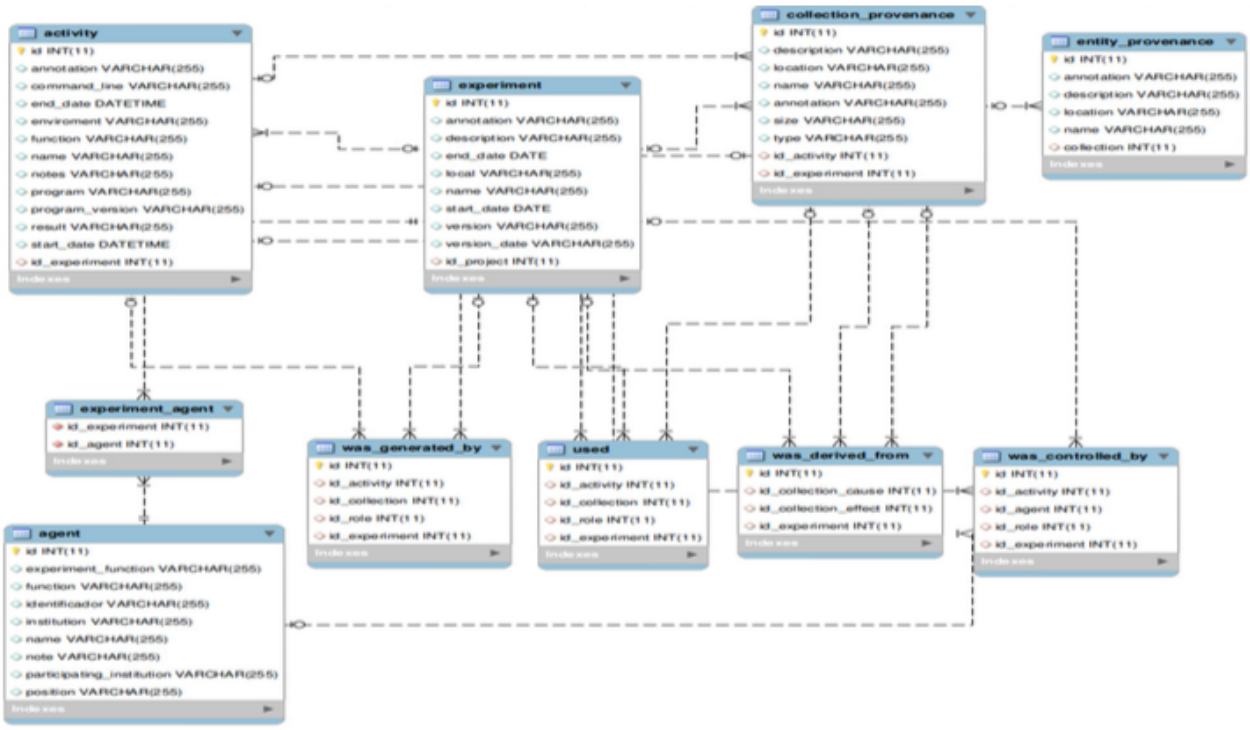

Figura 2.15: *Schema* relacional do modelo PROV-DM.

Baseando-se nesse modelo a consulta da Listagem 2.1 foi construída para recuperar todos os dados necessários para a montagem do grafo de proveniência. Observa-se a quantidade de *joins* necessários para obter os diversos tipos de relações existentes no modelo PROV-DM. A ideia da consulta é recuperar todas as atividades executadas em um experimento e a partir dessas atividades obter as relações envolvidas.

```
1  select a.name, a.command_line,
2         ag.identificador as WasControlledBy,
3         cu.name as Used, cg.name as WasGeneratedBy,
4         cp.name as origem,cp1.name as destino
5  from   db_provenance.experiment e
6  inner join db_provenance.activity a
7                on a.id_experiment = e.id
8         inner join db_provenance.was_controlled_by c
9                on c.id_activity = a.id
10        inner join db_provenance.agent ag
11               on ag.id = c.id_agent
12        inner join db_provenance.used u
13               on u.id_activity=a.id
14        inner join db_provenance.collection_provenance colprov
15               on colprov.id = u.id_collection
16        inner join db_provenance.was_generated_by w
17               on w.id_activity = a.id
18        inner join db_provenance.collection_provenance cg
19               on cg.id = w.id_collection
20        inner join db_provenance.collection_provenance
```

```
21              cp on cp.id_activity = a.id
22        left join db_provenance.was_derived_from d
23              on d.id_collection_cause = cp.id
24        inner join db_provenance.collection_provenance cp1
25              on cp1.id = d.id_collection_effect
26  where  e.id = 3
```

Listagem 2.1: Consulta para recuperar dados de proveniência

Já em Pinheiro et al. (2014) foi proposto um modelo de dados baseado em grafo. Um detalhamento do modelo proposto com as suas entidades e relacionamentos pode ser visto na Seção 3.2. A utilização do modelo baseado em grafos trouxe os seguintes benefícios:

- O modelo de proveniência PROV-DM é baseado em grafos, portanto usar uma estrutura de dados baseado em grafo torna a modelagem mais natural;
- A base de grafos foi desenvolvida para dados que possuem como principal característica uma grande quantidade de relações entre eles, sendo que os dados de proveniência possuem essa característica;
- A estrutura de dados de grafos são visíveis aos usuários, permitindo expressar consultas com um nível maior de abstração;
- Facilita a implementação de algoritmos eficientes para executar operações específicas.

Portanto, ao se analisar as consultas das listagens 4.2 e 2.1, observa-se a simplicidade de realizar *queries* no banco de dados de grafo em dados que envolve muitos relacionamentos. Outra característica observada é possuir uma *API* que facilita certas operações sobre o grafo, tal como a operação transversal.

2.7 Trabalhos Correlatos

Na literatura existem propostas de armazenamento de dados de proveniência de bioinformática, com o objetivo de tratar o modelo de dados, a forma de captura (que pode ser manual, semi-automática e automática) da proveniência e o sistema de armazenamento.

É proposto em de Paula et al. (2013) o uso de proveniência de dados com base no modelo PROV-DM para fluxos de trabalho de projetos genoma, permitindo aos cientistas estudar detalhes de suas experiências e, sempre que necessário re-executá-los de forma mais planejada e controlada. Para validação da proposta foi desenvolvido um simulador de proveniência para facilitar a inclusão e armazenamento dos dados. Observa-se que a proveniência de dados é feita manualmente, semelhante a um livro de registo, onde o cientista registra os dados durante a realização da experiência e os dados são salvos em arquivos *xml*.

Em Paulino et al. (2010), tem-se uma abordagem que apoia a coleta de metadados de proveniência em experimentos científicos, baseada na evolução da arquitetura *Matrioska* para o ambiente de nuvens. Além disso, também apresenta um modelo de dados capaz de armazenar os metadados de proveniência específicos da nuvem baseado no modelo *Open Provenance Model*. Para validação do trabalho foi aplicado como estudo de

caso um *workflow* de Mineração de Texto (MT) concebido e executado com o SGWfC (Sistema de Gerência de *Workflow*) *Vistrails* e como resultado conseguiu-se capturar um conjunto inicial de dados que não podiam ser coletados somente com os mecanismos locais de proveniência dos SGWfC, tais como, os identificadores de produtos de dados e a identificação das instâncias utilizadas (endereço IP) e o tipo de banco de dados e quais usuários executaram o *workflow*.

Uma abordagem desenvolvida na forma de um componente de software chamada *ReproeScience* para reprodução do ambiente onde o experimento computacional foi originalmente executado, de forma que o mesmo possa ser instanciado sob demanda e reproduzido em iguais condições foi proposto por de Oliveira et al. (2012). O *software* desenvolvido usa um modelo de dados baseados no modelo de proveniência PROV-DM. O *ReproeScience* coleta dados de proveniência por meio de sinais do sistema operacional, tais como interrupções e *system calls*, em seguida executa uma atividade de clonagem do ambiente original gerando imagens de máquinas virtuais prontas para serem reproduzidas em uma estação de trabalho ou na nuvem.

Devido a grande quantidade de dados gerados durante um experimento científico, pode ser inviável analisar os dados após a execução. Por isso, Costa et al. (2013) propõem uma solução para monitorar o experimento durante a sua execução usando dados de proveniência capturados de forma automática. A análise da proveniência em tempo de execução permitirá aos cientistas monitorar o *workflow* executado e tomar ações antes do seu final. O modelo de proveniência PROV-DM foi usado como base para a geração de um modelo relacional. Para a captura, foram desenvolvidos componentes que executam em paralelo com o *workflow* e importam os dados gerados pelos diferentes SGWfC para a base de dados chamada de PROV-Wf. Uma vez que todas as informações de proveniência importada dentro da base de dados, consultas podem ser realizadas em tempo de execução para realizar a análise do experimento e assim sejam tomadas medidas de controle caso algum erro ocorra.

Em um cenário onde a execução de *workflow* produz uma grande quantidade de dados, sua execução pode levar dias ou semanas, até mesmo em ambientes de computação de alto desempenho. Nesse contexto, uma forma de melhorar a performance é reduzir os dados a serem processados, ou seja, estabelecer eventos que possam determinar se os dados produzidos são válidos ou não para serem consumidos pela próxima atividade no *workflow*. Em Gonçalves et al. (2013) é proposto eliminar dados intermediários com pouca qualidade analisandos dados de proveniência em tempo de execução do *workflow*. Para isso foi desenvolvido um componente de software, chamado *Provenance Analyzer* que permite extrair dados de proveniência de arquivos intermediários e filtrar esses dados baseados em critérios também fornecidos. Para a validação da proposta, foram executado diversos experimentos na nuvem da *Amazon*, obtendo resultados que melhoraram o tempo de execução do *workflow* em até 36.2%.

A execução de projetos de bioinformática necessita de uma variedade de processos que podem ser executados com diferentes opções de programas e diversas formas de uso através de parâmetros e configurações. Portanto, em Pinheiro et al. (2013) é proposta a captura dos dados de proveniência semi-automática e a definição de um esquema relacional baseado no modelo PROV-DM com o objetivo de facilitar a validação e publicação do experimento. Como principal resultado conseguiu-se gerar o grafo de proveniência da execução de *workflows* em Bioinformática.

Conforme Woodman et al. (2011), uma descrição de como um sistema de armazenamento de proveniência é usado pelo *e-Science Central* que pode ser usado para responder questões importantes para a pesquisa científica. Como por exemplo, qual a proveniência de uma peça de dado e quais os efeitos ao mudar as versões de um artefato? Aproveitando a estrutura de armazenamento de proveniência e controle de versão disponibilizados pelo *e-Science Central* é possível responder a essas questões, assim como executar o *workflow* em uma versão antiga e comparar os resultados com versões mais recentes. O modelo de proveniência usado é o OPM, que foi implementado com um banco de dados de grafo, *Neo4J*, devido a facilidade de realizar consultas transversais sobre a estrutura do grafo.

Em Korolev and Joshi (2014) usa uma ferramenta de captura de dados de proveniência para alcançar a reprodução de experimentos científicos envolvidos com *workflow* de *big data* chamada PROB. A ferramenta é baseada no *Git*, *Git-Annex* e *Git2Prov*. O *Git* é usado como um sistema de controle de versão e um repositório dos dados de proveniência. O *Git-Annex* é uma extensão do *Git* que permite rastrear informações de versão de grandes arquivos sem buscá-los dentro do repositório. O *Git2Prov* é uma ferramenta para converter os metadados armazenados no *Git* para o formato PROV-DM. Além disso, o PROB define sua própria ontologia para representar a informação de proveniência, que foi baseada na ontologia do *Git2Prov* e do *NiPype*, que é usado para expressar fatos sobre análise de dados dos *workflows* e ambientes de execução.

A maioria das ferramentas disponíveis coletam apenas parte da proveniência (a que é explicitada na especificação do *worflow*), levando a perda de informações importantes para a análise do experimento. Uma solução proposta pode ser encontrada em Oliveira et al. (2014) que apresenta uma abordagem para coleta, armazenamento e consulta da proveniência obtida pelo monitoramento dos diretórios e arquivos manipulados pelas atividades do *workflow*. Logo informações não referenciadas na especificação do *workflow* são coletadas e relacionadas com a proveniência previamente especificada. Foi desenvolvido um módulo chamado *ProvWatcher* que monitora e recupera informações dos diretórios onde os dados se encontram e as armazena em um banco de dados relacional. Para validação da proposta foi executado um estudo de caso utilizando o mecanismo de execução de *workflow* denominado SciCumulus, que obteve um resultado satisfatório devido aos dados gerados pelo monitoramento do *ProvWatcher* permitirem a realização das consultas propostas possibilitando a validação do experimento executado.

Uma outra aplicação da proveniência de dados é identificada em de Oliveira et al. (2014), que propõe uma forma de realizar o *debugging* do *workflow* em tempo de execução, usando o modelo PROV-DM beneficiando-se da redução da incidência de erros, da diminuição do tempo total de execução e do custo financeiro. Foi utilizado uma técnica baseada em álgebra para identificação dos dados, usando as relações de proveniência prospectiva para armazenar a especificação do *workflow* e as relações retrospectiva para armazenar a execução do *workflow*. Esta técnica introduz um conjunto de operadores para a álgebra relacional (*Map*, *Reduce*, *Filter*, *SplitMap*, *MRQuery* e *SRQuery*) que considera tanto as atividades como os dados do fluxo como operandos. Para validação da proposta foram executados dois estudos de caso, o qual demonstrou redução de tempo na execução dos *workflows* devido a identificação de erros precocimente identificados através de consultas executadas em tempo de execução.

A seguir tem-se a Tabela 2.7, que resume o referencial teórico em termos do ambiente de execução, modelo de proveniência e sistema de armazenamento utilizado. Como pode

ser observado a maioria dos modelos de proveniência usados são OPM ou PROV-DM, com alguns trabalhos na literatura utilizando bancos de dados relacional e de grafos para armazenar dados de proveniência.

Tabela 2.2: Resumo Referencial Teórico.

Artigo	**Modelo de Proveniência**	**Sistema de armazenamento**	**Captura de proveniência**
de Paula et al. (2013)	Arquivos XML	Manual	
Paulino et al. (2010)	OPM	Banco de dados relacional	Automática
de Oliveira et al. (2012)	PROV-DM	Banco de dados relacional	Automática
Costa et al. (2013)	PROV-DM	Banco de dados relacional	Automática
Gonçalves et al. (2013)	—	Banco de dados relacional	—
Pinheiro et al. (2013)	PROV-DM	Banco de dados relacional	Automática
Woodman et al. (2011)	OPM	Banco de dados de grafo	—
Korolev and Joshi (2014)	PROV-DM	Sistema de versionamento *GitHub*	—
Oliveira et al. (2014)	PROV-DM	Banco de dados relacional	Automática
de Oliveira et al. (2014)	PROV-DM	Banco de dados relacional	—
Arquitetura proposta	PROV-DM	Banco de dados de grafo	Automática

Capítulo 3

Arquitetura Proposta

Este capítulo apresenta a arquitetura proposta para armazenamento de dados de proveniência, sendo dividido em três seções, a primeira realiza uma contextualização da proposta, a segunda o mapeamento das entidades e relacionamentos do modelo PROV-DM para o modelo de um banco de dados de grafo e a terceira especifica a arquitetura de armazenamento de proveniência.

3.1 Contextualização da Proposta

A proposta trata de um modelo de dados para proveniência de *workflow* de Bioinformática usando bancos de dados de grafos para armazenar os dados do modelo PROV-DM com automatização da captura de proveniênica. Neste contexto, os principais aspectos que levaram a definição desta arquitetura são:

- Como apresentado em de Paula et al. (2013), o modelo PROV-DM pode ser aplicado em *workflow* de Bioinformática onde através de um grafo é possível facilmente representar a proveniência em um experimento da Bioinformática;
- Como o PROV-DM é um modelo baseado em grafo, onde toda proveniência pode ser representada através de nós e arestas, este trabalho propõe como contribuição o armazenamento dos dados de proveniência em um banco de dados de grafo;
- Conforme pode ser visto na Seção 2.7, a captura dos dados de proveniência de forma automática não é uma realidade em pequenos experimentos na Bioinformática no qual não utiliza Sistemas de Gerência de *Workflow*.

Sendo assim, a proposta apresenta uma arquitetura capaz de realizar a proveniência automática do tipo prospectiva (já que captura os passos que devem ser seguidos para a geração de um dado produto), retrospectiva (pois captura informações obtidas durante a execução dos processos de geração do dado, por exemplo a duração de cada atividade executada) e dados definidos pelo usuário (informações como funções e versões dos programas utilizados). Quanto ao nível de captura realizado é *workflow* (pois realiza a captura das atividades que fazem parte do *workflow*) e atividade (porque captura dados referentes a execução dos processos). O contexto utilizado é em experimentos de Bioinformática utilizando um modelo de armazenamento baseado em grafos.

3.2 Modelo de Dados de Proveniência em Bancos de Dados de Grafo

Banco de dados de grafos permitem o armazenamento de entidades e relacionamentos entre essas entidades, que também são conhecidas como nós, os quais possuem propriedades. Os relacionamentos são conhecidos como arestas que também podem ter propriedades. As arestas têm significância direcional; nós comunicam-se por relacionamentos, os quais permitem encontrar padrões entre eles.

Sendo assim os três tipos básicos do modelo PROV-DM, atividade, agente e entidade são representados como nós no banco de dados de grafo, e são diferenciados pela propriedade Tipo. Já as relações são representadas pelas arestas no banco de dados de grafo, sendo diferenciadas também por uma propriedade Tipo. Na Figura 3.1 tem-se o mapeamento mais detalhado dos tipos de entidades do modelo PROV-DM. Para a construção do modelo de dados em grafo foi escolhido o modelo baseado em atributos porque se beneficia de certos tipos de consultas que realizam buscas sobre atributos.

Para uma definição mais detalhada dos componentes da Figura 3.1, tem-se os seguintes nós:

- Experimento: representa um conjunto de informações (tipos e relações) que compõe um grafo de proveniência. É tratado como a execução de um experimento, portanto representa o próprio grafo de proveniência;
- Atividade: representa os possíveis processos executados em um experimento. Este elemento indicará as propriedades do programa executado, incluindo linhas de comando, nome do programa, versão, entre outros. Vale ressaltar que, dos elementos que compõe o grafo(coleção, agente, grupo e atividade), este é o único que possui características temporais (hora inicial e hora final);
- Coleção: representa um arquivo utilizado ou gerado durante a execução de uma atividade;
- Agente: representa qualquer entidade (pessoa, organização, software, serviço, etc...) que possa ter algum tipo de ação sobre uma Atividade ou possa ter algum tipo de responsabilidade sobre uma Coleção;
- Grupo: a fim de propiciar a criação de grafos de proveniência personalizados, a partir dos experimentos executados, foi definido um atributo chamado Grupo. Este atributo está presente nos possíveis nós do grafo, ou seja, os elementos coleção, agente e atividade. Os grupos são definidos pelo usuário durante o cadastramento de cada elemento e, por ser um atributo múltiplo, cada elemento pode fazer parte de diferentes grupos. Dessa forma o usuário poderá solicitar a visualização de parte do grafo escolhendo os grupos desejados. Esse recurso é especialmente importante para grafos muito grandes onde o usuário pode querer visualizar somente parte dele.

Já quanto as arestas/relacionamentos tem-se:

- *Used*: Esta relação liga uma coleção ou uma entidade a atividade que está usando esta coleção. Como uma aresta direcionada ela sempre vai da atividade para a coleção. Cada coleção ou entidade poderá ser usada em diversas atividades. Da mesma

forma que cada atividade poderá usar diversas coleções. Isso está de acordo com os experimentos executados em projetos de bioinformática onde um determinado arquivo pode ser usado por diferentes processos, como no caso de um genoma de referência, por exemplo, que pode ser utilizado para o alinhamento com diferentes arquivos com *reads* em diferentes processos de alinhamento. Também um processo pode utilizar vários arquivos, como um processo de mesclagem por exemplo, onde dois ou mais arquivos com *reads* podem ser unidos formando um arquivo único.

- *WasAssociatedWith*: indica que um agente teve algum tipo de ação sobre uma atividade. Seu direcionamento parte da atividade para o agente. Também são permitidas múltiplas ligações entre diferentes agentes e atividades.
- *WasDerivedFrom*: indica a ligação de derivação, durante a execução do experimento, entre os dados utilizados e gerados. Assim como os demais tipos de relacionamento, a aresta que a relação *wasDerivedFrom* é direcionada e aponta da coleção derivada para a respectiva coleção original. Também são permitidas múltiplas ligações entre coleções, já que uma coleção pode ser derivada de diversas outras coleções, assim como uma coleção pode ser utilizada na geração de diversas outras coleções ou entidades.
- *WasGeneratedBy*: indica qual atividade gerou uma determinada coleção. Seu direcionamento, como aresta, parte da coleção para a atividade que as gerou. Uma atividade poderá gerar diversas coleções ou atividades, porém cada coleção somente pode ser gerada por uma única atividade. Dessa forma diversas relações WasGeneratedBy poderão ser feitas para uma atividade, porém somente uma poderá ser feita para cada coleção ou entidade. Essa definição corresponde aos processos executados em projetos de bioinformática, onde um determinado dado (ou arquivo) somente pode ter sido gerado pela execução de um único programa.

Foi-se necessário o acréscimo da entidade Projeto que não pertence ao modelo PROV-DM, devido a necessidade de agrupar experimentos, pois na Bioinformática um projeto pode envolver a execução de diversos experimentos para se chegar a uma conclusão, conforme simplificado na Figura 3.2. Portanto, usuários podem separar os experimentos em diferentes grupos e dividí-los. Em outras palavras, um projeto é um *container* que permite aos usuários agrupar os experimentos.

Já o tipo *Account* (Experimento) não possui símbolos no modelo PROV-DM porque representa o próprio grafo de proveniência.

3.3 Arquitetura Proposta

Esta seção trata da arquitetura para a coleta de proveniência de dados em projetos de bioinformática de forma automática usando o modelo de dados baseado em grafo definido na Seção 3.2.

3.3.1 MVC - *Model-View-Controller*

O padrão arquitetural *Model-View-Controller* (MVC) é uma forma de quebrar uma aplicação em três partes: o modelo, a visão e o controlador.

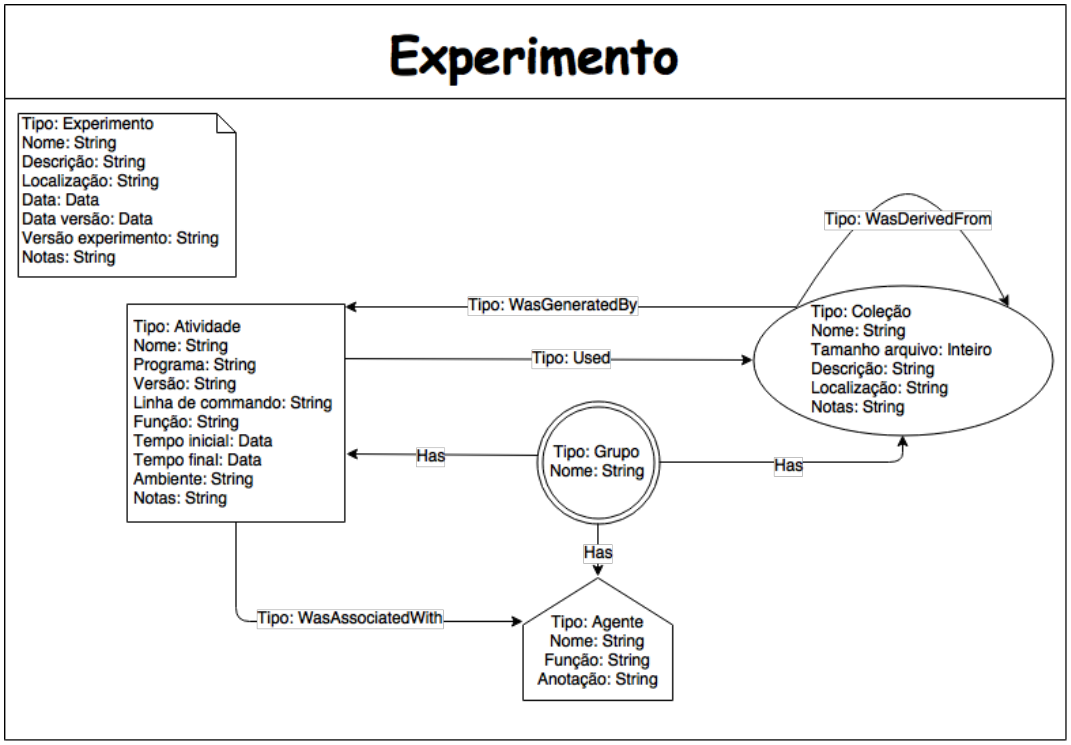

Figura 3.1: Mapeamento das entidades PROV-DM para o modelo de grafo de atributos.

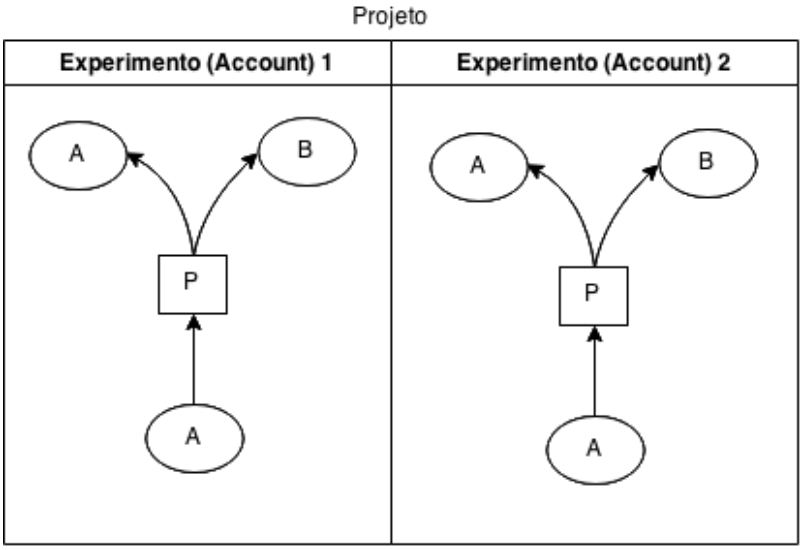

Figura 3.2: Um projeto agrupa diferentes experimentos executados.

A Figura 3.3 exibe os componentes do MVC e a iteração entre eles. O controlador (*Controller*) interpreta as entradas da interface enviadas pelo usuário e mapeia essas ações em comandos que são enviados para o modelo (*Model*) e/ou para a janela de visualização (*View*) para efetuar a alteração apropriada. Por sua vez o modelo gerencia um ou mais elementos de dados, responde a perguntas e a mudanças sobre o seu estado. Por fim, a visão é responsável por apresentar as informações para o usuário através de uma com-

binação de gráficos e textos. A visão não sabe sobre o que a aplicação está atualmente fazendo, o que ela faz é receber instruções do controle e informações do modelo e então exibí-las.

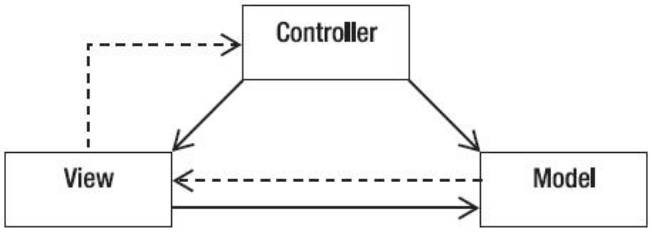

Figura 3.3: Objetos utilizados no MVC e suas interações.

Portanto, a principal ideia do padrão arquitetural MVC é a separação dos conceitos - e do código. Entre as diversas vantagens do padrão MVC estão a possibilidade de reescrita da GUI ou do Controller sem alterar o nosso modelo, reutilização da GUI para diferentes aplicações com pouco esforço, facilidade na manutenção e adição de recursos, reaproveitamento de código, facilidade de manter o código sempre limpo, etc.

3.3.2 Definição dos módulos

Na Figura 3.4 pode-se ver os principais componentes da arquitetura, que estão definidos a seguir:

- Uma interface web simples e amigável para o usuário, permitindo-o criar projetos, experimentos e atividades. Os dados informados pelo usuário serão enviados para o servidor para serem processados;
- Controller é o módulo responsável apenas por receber a requisição do usuário, realizar algumas validações e delegá-las para o módulo de processamento;
- Módulo de processamento realiza o processamento das requisições, tais como executar a regra de cadastro de projetos, experimentos, atividades, separar os comandos a serem executados e enviar para uma fila;
- A fila possui o conjunto de comandos a serem executados que foram enviados pelos usuários. A sua criação foi necessária para garantir que todos os comandos do usuário sejam executados e devido a necessidade desses comandos serem executados de forma *batch*;
- Módulo de execução é responsável por executar os comandos recebidos do usuário. Ele recupera o comando da fila e cria os diretório para a monitoração pelo módulo *Watcher*;
- *Watcher* realiza o monitoramento do diretório raiz criado para a execução de um comando da atividade, pela coleta da proveniência retrospectiva, ou seja, arquivos criados, tamanho dos arquivos e data de criação;
- Módulo de armazenamento provê uma *API* de comunicação com o banco de dados de grafo, realizando operações de inserção, deleção, atualização e consultas;

- Banco de dados de grafo é responsável pelo armazenamento dos dados de proveniência na forma de grafo.

A execução de um programa pelo simulador *Provenance* se dá a partir do momento que o usuário cadastra uma atividade. Ao submeter uma requisição, o módulo *Controller* realiza algumas validações de obrigatoriedade e passa para o módulo de processamento uma ordem de execução de atividade. Nesse momento, o módulo de processamento solicita ao módulo de armazenamento a criação de uma atividade com os parâmetros informados no banco de dados de grafo, simultaneamente, o módulo de processamento coloca na fila o comando de execução informado e realiza um retorno de sucesso para o usuário. Enquanto isso, o módulo de execução recupera o comando da fila, cria a pasta de saída de execução e um serviço de *Watcher* (responsável pelo monitoramento da pasta de saída) que atualizará/armazenará informações referente as coleções (tais como arquivos gerados, tamanhos de arquivos, hora, etc) no banco de dados de grafo.

A seguir cada um dos componentes são descritos com mais detalhes.

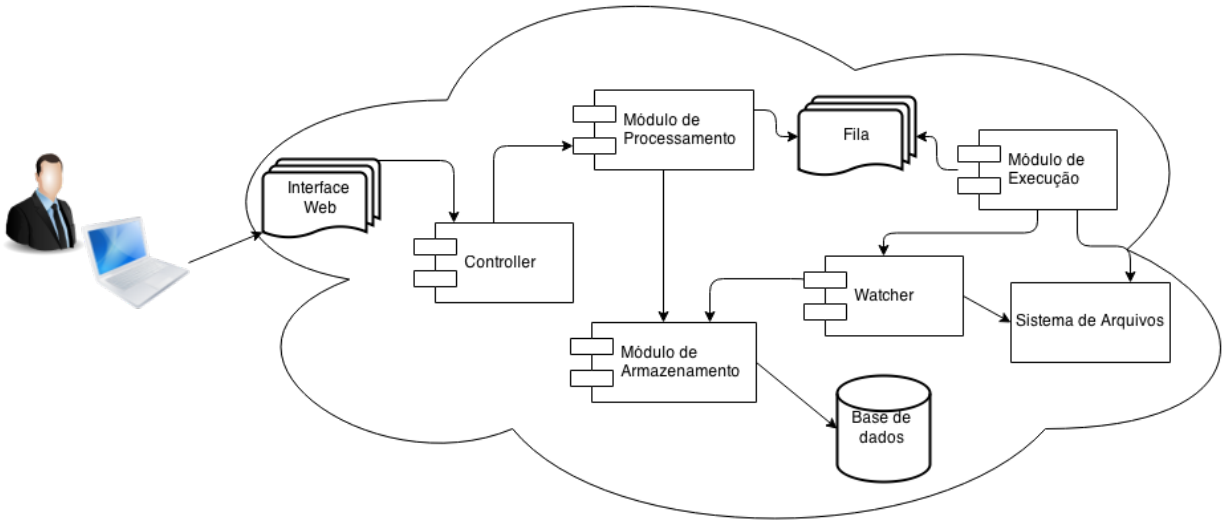

Figura 3.4: Arquitetura Proposta.

3.3.3 Interface Web

Para o usuário acessar a arquitetura proposta foi implementada uma interface web simples, visando sempre uma melhor usabilidade. Foi desenvolvida utilizada as tecnologias *HTML5*, *JSP* (*Java Server Pages*), *JavaScript* e *jQuery*. Os critérios usados para a seleção das tecnologias foram facilidade de desenvolvimento, quantidade de documentação disponível e tecnologias gratuitas.

A Figura 3.5 corresponde as telas de *login* e cadastro de usuários respectivamente. Caso o usuário não possua cadastro, ele pode realizar o seu próprio cadastro clicando no *link* 'Não tenho usuário, quero me cadastrar'. E logo em seguida, realizar *login* no sistema.

Figura 3.5: Telas de login e cadastro de usuários.

Já a Figura 3.6 trata da tela principal do sistema. Nela pode-se ver à esquerda a estrutura de árvore que as entidades da proveniência (projeto, experimento e atividade) estão organizadas. Ao se clicar em uma das entidades da árvore é exibido os seus respectivos detalhes. No topo encontra-se um menu onde é possível criar novas instâncias das entidades.

3.3.4 *Controller*

O *Controller* é responsável por enviar comandos para a sua visão associada para alterar a apresentação da visão do modelo (por exemplo, ao clicar na entidade projeto o *Controller* enviará os dados da entidade para a tela). Ele também envia os comandos para o modelo para atualizar o estado (por exemplo, ao savar um projeto).

Na Figura 3.7 tem-se os seguintes serviços responsáveis por realizarem a comunicação da *view* com o módulo de processamento:

- *ProjectController*: gerencia a comunicação entre a interface e o módulo de processamento para as operações de inclusão, alteração, remoção e visualização da entidade Projeto (*Project*).

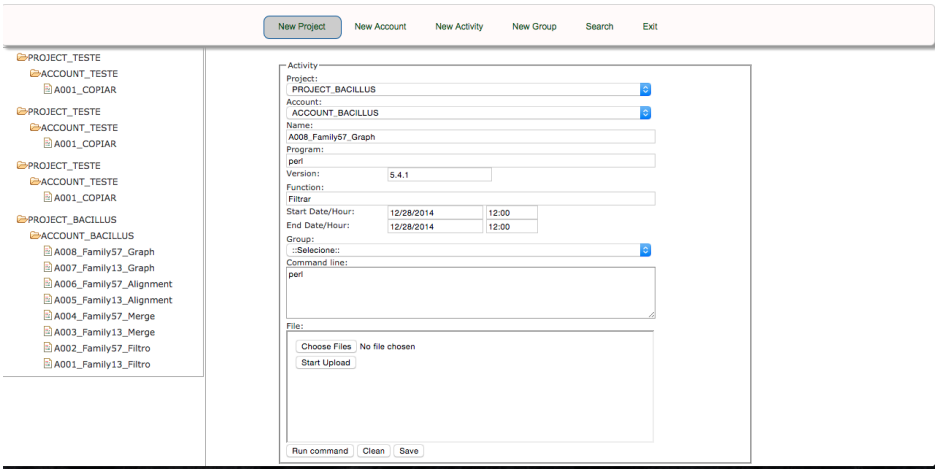

Figura 3.6: Tela principal do sistema.

- *AccountController*: gerencia a comunicação entre a interface e o módulo de processamento para as operações de inclusão, alteração, remoção e visualização da entidade Experimento (*Account*).
- *ActivityController*: gerencia a comunicação entre a interface e o módulo de processamento para as operações de inclusão, alteração, remoção e visualização da entidade Atividade (*Activity*).
- *CollectionProvenanceController*: gerencia a comunicação entre a interface e o módulo de processamento para as operações de *upload* e visualização da entidade Coleção (*Collection*).
- *GroupController*: gerencia a comunicação entre a interface e o módulo de processamento para as operações de inclusão, alteração, remoção e visualização da entidade Grupo (*Group*).
- *UserController*: gerencia a comunicação entre a interface e o módulo de processamento para as operações de inclusão, alteração, remoção e visualização da entidade Usuário (*User*).

3.3.5 Módulo de Processamento

O módulo de processamento define a fronteira de uma aplicação e o seu conjunto de operações disponíveis a partir da perspectiva da interface cliente. Ele encapsula a lógica de negócio, controlando as transações e coordenando as respostas na implementação das operações.

Para a construção do simulador foram necessários os serviços da Figura 3.8 e definidos a seguir:

- *ProjectService*: é responsável pelo processamento da entidade Projeto (*Project*), tais como métodos para salvar, atualizar e buscar projetos.
- *AccountService*: é responsável pelo processamento da entidade Experimento (*Account*), tais como métodos para salvar, atualizar e buscar experimentos.

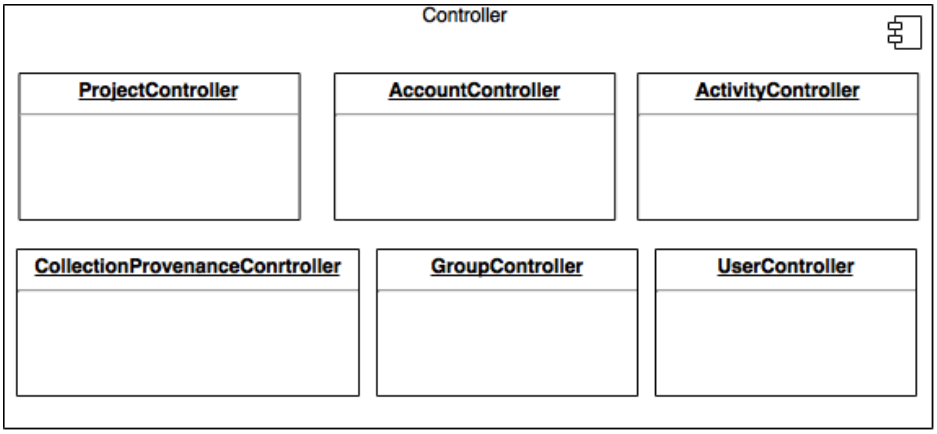

Figura 3.7: Diagrama de serviços da camada de processamento.

- *ActivityService*: é responsável pelo processamento da entidade Atividade (*Activity*), tais como métodos para salvar, atualizar e buscar atividades.
- *CollectionProvenanceService*: é responsável pelo processamento da entidade Coleção (*Collection*), tais como métodos para salvar, atualizar, buscar coleções e arquivos.
- *GroupService*: é responsável pelo processamento da entidade Grupo (*Group*), tais como métodos para salvar, atualizar, buscar e associar grupos a atividades.
- *UserService*: é responsável pelo processamento da entidade Usuário (*User*), tais como métodos para salvar, atualizar e buscar usuários.

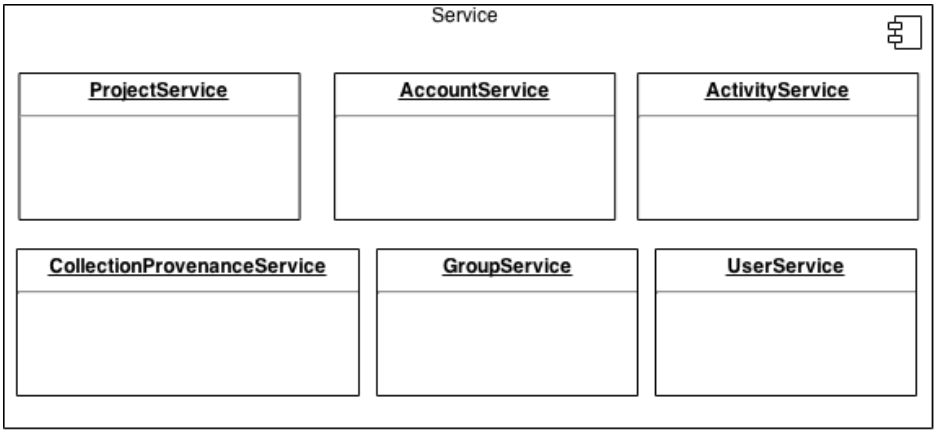

Figura 3.8: Diagrama de serviços da camada de processamento.

3.3.6 Módulo de Armazenamento

Este módulo implementa o padrão *repository*, que representa todos os objetos de um certo tipo como um conjunto conceitual (Evans, 2004). Este padrão de desenvolvimento atua como uma coleção, exceto que com uma capacidade de consulta mais elaborada. Objetos são adicionados ou removidos, e o mecanismo por trás do padrão *repository* irá

inserí-los ou removê-los do banco de dados. Esta definição reúne um conjunto coeso de responsabilidades para garantir o acesso aos dados.

Os serviços classes que compõem o módulo de armazenamento podem ser observadas na Figura 3.9 e definidos a seguir:

- *ProjectRepository*: expõe operações para manipulação dos dados do projeto.
- *AccountRepository*: expõe operações para manipulação dos dados do experimento.
- *ActivityRepository*: expõe operações para manipulação dos dados da atividade.
- *CollectionProvenanceRepository*: expõe operações para manipulação dos dados da coleção ou arquivo.
- *GroupRepository*: expõe operações para manipulação dos dados do grupo.
- *UserRepository*: expõe operações para manipulação dos dados do usuário.

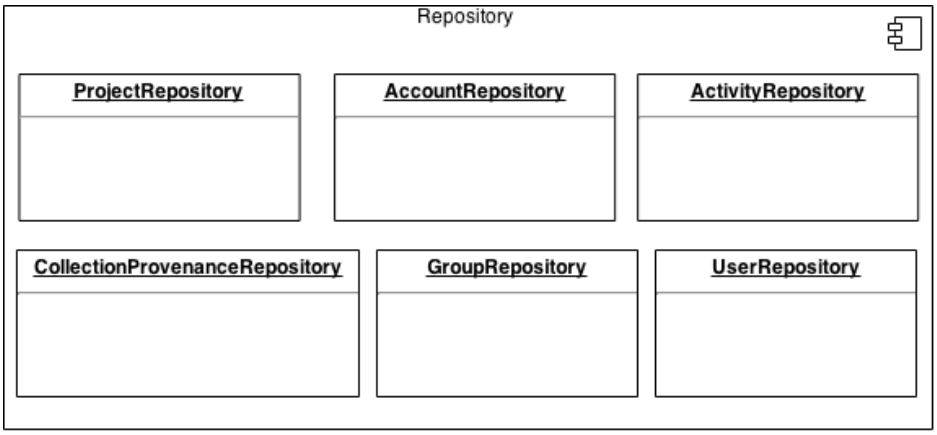

Figura 3.9: Diagrama de serviços da camada de armazenamento.

3.3.7 Fila

De acordo com Cormen (2002) uma fila é uma estrutura de dados que permite que o elemento eliminado é sempre o que esteve no conjunto pelo tempo mais longo, ou seja, a fila implementa uma norma de prioridade de primeiro a entrar, primeiro a sair, ou *FIFO* (*first in, first out*).

Dentro da especificação *JEE* (*Java Enterprise Edition*), existe uma API voltada ao serviço de mensageria, chamada de *JMS* (*Java Message Service*). Através dessa API as aplicações desenvolvidas em Java se comunicam com o servidor *JMS*, que é independente do fabricante de MOM. Sendo que MOM (*Middleware Oriented Message*) é um software que realiza a comunicação entre as aplicações, por exemplo *CORBA*, *Enterprise Service Bus* significando dizer que uma vez que se trabalhe com a API, espera-se que ela funcione com qualquer MOM do mercado.

Com o uso do *JMS* para integração das aplicações pode-se conseguir um nível satisfatório no que tange ao baixo acoplamento entre as aplicações, uma vez que as mesmas não

precisam tomar conhecimento da outra aplicação a qual ela irá se comunicar. Seu funcionamento é bastante simples, consiste de um componente enviar uma mensagem para um destinatário e o destinatário irá retirá-la do repositório, no caso o MOM. Conforme pode ser visto na Figura 3.10, o Cliente 1 envia uma mensagem para a fila, esta por sua vez é retirada e processada pelo Cliente 2.

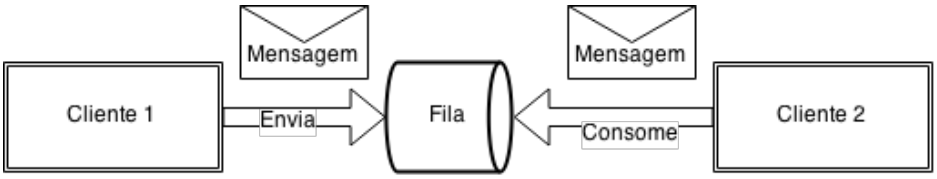

Figura 3.10: Funcionamento de uma fila.

3.3.8 Watcher

O *Watcher* é responsável pelo monitoramento do diretório de execução e pela coleta de dados de proveniência relacionados aos arquivos de entrada e saída usados na execução de processo. Foi desenvolvido na linguagem *Java* devido ao fato de ser multiplataforma e possuir uma API própria para notificação de mudanças de arquivos denominada *WatchService* (Oracle, 2015).

A API *Watcher* opera sobre a *JVM* (*Java Virtual Machine*) fornecendo uma abstração do sistema de arquivos e do sistema operacional utilizado, com o intuito de facilitar a portabilidade da aplicação entre diferentes sistemas operacionais. Conforme a Figura 3.11, o *Watcher* realiza o monitoramento do diretório. Para cada novo arquivo criado no diretório, uma chamada ao módulo de armazenamento é realizada, que por sua vez recupera as informações do arquivo e as atualiza na base de dados de grafo.

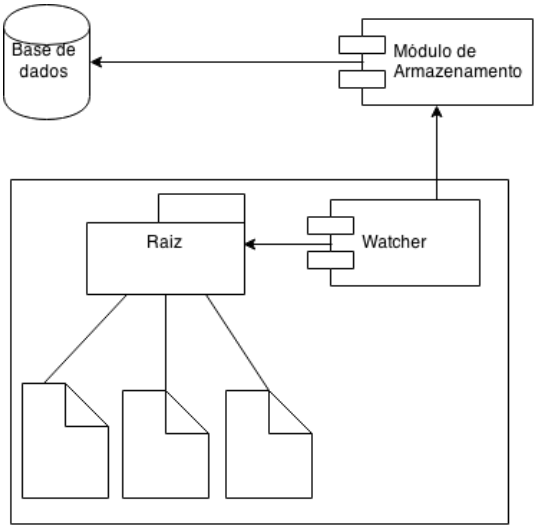

Figura 3.11: Funcionamento do módulo *Watcher*

3.4 Bancos de Dados de Grafo - *Neo4J*

O modelo de dados de grafo é uma estrutura na qual o esquema e/ou instâncias são modeladas como um grafo dirigido, podendo ser rotulados ou não. O modelo de grafos usado na arquitetura foi o de atributos, pois permitem que tanto vértices como as arestas armazenem informações de atributos como um par de chave e valor. Essa característica permite que certas informações sejam armazenadas próximas aos nós, facilitando a recuperação do dado.

O banco de dados de grafo utilizado para implementação do modelo proposto na Seção 3.2 foi o *Neo4J* porque permite o armazenamento do modelo de grafo de atributos, possui uma boa documentação disponível, funciona de forma embarcada na aplicação e é simples de configurar e usar.

Pode-se encontrar em Partner et al. (2013), que o *Neo4J* possui a linguagem de consulta *Cypher* específica de domínio para percorrer grafos. Esta linguagem precisa de um nó inicial (cláusula *Start*) para iniciar a consulta. Outra funcionalidade bastante utilizada na linguagem é a de buscar padrões em relacionamentos através da palavra-chave *Match*. *Cypher* ainda permite filtrar as propriedades em um nó ou relacionamento utilizando a cláusula *Where* e especificar o que deve ser retornado pela consulta com a palavra-chave *Return*.

Capítulo 4

Estudos de Caso

Este capítulo descreve o simulador *Provenance* e a sua aplicação em dois estudos de caso de projetos de bioinformática. O capítulo está dividido da seguinte forma: a Seção 4.1 descreve as principais características do simulador *Provenance*, na Seção 4.2 é demonstrado a aplicação do simulador em dois estudos de caso com dados reais.

4.1 Simulador *Provenance*

O simulador *Provenance* foi implementado com o objetivo de demonstrar a validade da arquitetura proposta no Capítulo 3.3 para *workflows* de bioinformática. A construção deste simulador foi baseada em algumas premissas:

- Implementar as características do modelo de proveniência de dados PROV-DM para *workflows* de bioinformática;
- Criar uma interface simples e amigável para o usuário;
- Facilitar o intercâmbio de informações entre diferentes equipes;
- Realizar a captura dos dados de proveniência de forma automática.

O simulador desenvolvido permite a captura de forma automática em *workflows* de Bioinformática e o armazenamento dos dados de proveniência em bancos de dados de grafo. Para a implementação do simulador, optou-se por uma versão web que utiliza a linguagem Java, como já está especificado no Capítulo 3.3. Como linguagem de apresentação utilizou-se *JSP*, *JavaScript* e *jQuery* (jQuery, 2015) devido a facilidade de aprendizagem, uso bastante difundido no mercado e com uma documetação bem consolidada. Para gerenciamento e comunicação entre as camadas utilizou-se o *framework VRaptor* (Vraptor, 2015). E finalmente, quanto a persistência foi utilizado o banco de grafos *Neo4J*.

4.1.1 Ambiente Computacional

Amazon EC2 (*Amazon Elastic Compute Cloud*) é um dos serviços mais utilizados da plataforma de computação em nuvem da *Amazon* (*AWS - Amazon Web Services*) e permite criar instâncias de servidores virtuais na nuvem com variadas configurações de

sistemas operacionais como *Linux* e *Windows*, assim como diferentes configurações de processador, memória e armazenamento de disco.

Na AWS um servidor virtual que executa na nuvem é conhecido como instância. No EC2, o poder dos processadores é medido numa unidade conhecida como *EC2 Compute Unit* (ECU). Uma ECU fornece a capacidade de CPU equivalente de um processador *Opteron 2007* ou *Xeon 2007* de 1.0 a 1.2 GHz (Lecheta, 2014). Por exemplo, uma instância do tipo *micro* pode executar até duas unidades de processamento EC2 (ECU), possui 613 MB de memória e baixa velocidade de rede. Existem vários tipos de instâncias no EC2, instâncias de uso geral com as mais variadas configurações, instâncias otimizadas para computação com alta capacidade de processamento e instâncias otimizadas para memória ou armazenamento.

O simulador *Provenance* foi instalado e configurado em uma máquina virtual da *Amazon EC2* com as seguintes configurações:

- Instância t2.small;
- Ubuntu Server 14.04 LTS, 64 bits;
- 50GB de HD;
- 8 GB (*gigabyte*) de memória;
- Processadores *Intel Xeon* de alta frequência, operando a 2,5 GHz com Turbo até 3,3 GHz.

Para que o simulador ficasse disponível na internet foi necessário habilitar o acesso utilizando o *Amazon Route 53* como serviço de DNS. Para hospedagem o site do simulador *Provenance* foi comprado o domínio *http://www.provenance.com.br/provenance* e utilizado o serviço responsável por traduzir nomes de domínios para endereços IP.

Além da configuração desses serviços, realizou-se a instalação e configuração de um servidor de aplicação (*Applications Server*), que é um servidor no qual disponibiliza um ambiente para a instalação e execução de certas aplicações, centralizando e dispensando a instalação nos computadores clientes. Os servidores de aplicação também são conhecidos por *middleware*.

O objetivo do servidor de aplicações é disponibilizar uma plataforma que separe do desenvolvimento de software algumas das complexidades de um sistema computacional, tais como questõs de infraestrutura da aplicação. O servidor de aplicações responde a algumas questões comuns a todas as aplicações, como segurança, garantia de disponibilidade, balanceamento de carga e tratamento de exceções.

Devido a popularização da plataforma Java, existem várias implementações de servidores, tais como: *IBM WebSphere Application Server* (WebSphere, 2015), *Oracle Oracle9i Application Server* (Oracle9i, 2015), *Wildfly* (Wildfly, 2015) e *Apache Tomcat* (Tomcat, 2015). O servidor de aplicação selecionado foi o *Wildfly* da *Red Hat JBoss*, por ser totalmente desenvolvido em *Java*, multiplataforma, possuir código aberto, ampla documentação e estar consolidado no mercado.

4.1.2 Bibliotecas Externas

Para a construção do simulador foram utilizadas as bibliotecas externas *Vraptor* (Vraptor, 2015), *jQuery* (jQuery, 2015) e *Viz* (Viz, 2015), descritas a seguir.

O *VRaptor* é um *framework* MVC desenvolvido em Java, focado no desenvolvimento rápido, simples e na fácil manutenção do código. O seu objetivo principal é realizar a mediação da interface, convertendo-a em comandos para o modelo ou visão.

jQuery é uma biblioteca *JavaScript* desenvolvida para simplificar os scripts *client side* que interagem com o HTML. Ela foi lançada em dezembro de 2006 no *BarCamp* de *Nova York* por John Resig. Usada por cerca de 77% dos 10 mil sites mais visitados do mundo, *jQuery* é a mais popular das bibliotecas JavaScript (jQuery, 2015).

A *Viz.js* é uma biblioteca *JavaScript* para o *GraphViz* (*Graph Visualization*), que é um pacote que fornece diversas ferramentas para tratamento de grafos em diferentes formatos. Um dos recursos fornecidos pelo pacote é processar arquivos no formato DOT Ellson et al. (2004) criando uma imagem do grafo, que pode ser gravado no formato GIF. DOT uma linguagem textual que permite a especificação de diferentes tipos de diagramas. Conforme pode ser visto na Figura 4.1, tem-se o código que gerou o grafo representado. No código, as linhas 2 à 5 definem os quatro nós do grafo enquanto que as linhas 6 à 9 definem as arestas, a linha 10 determina que os nós a (Coleção A) e b (Coleção B) fiquem na mesma altura no grafo.

```
1.   digraph graphexemplo {
2.         a [label="Coleção A"];
3.         b [label="Coleção B"];
4.         c [label="Atividade Filtragem", shape=box];
5.         d [label="Coleção Filtrada", shape=house];
6.         a -> c -> d;
7.         b -> c;
8.         d -> a [style = dotted];
9.         d -> b [style = dotted];
10.        {rank=same; a b}
11.  }
```

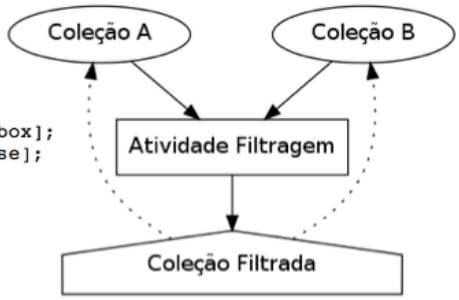

Figura 4.1: Exemplo de um grafo construído a partir da linguagem DOT.

4.1.3 Banco de Dados de Grafo - *Neo4J*

O sistema gerenciador de banco de dados utilizado foi o *Neo4J* que provê uma API (*Application Programing Interface*) para realizar consultas. Baseado nisto, foi criado o algoritmo da Listagem 4.1 que realiza uma operação transversal para carregar os dados de proveniência de um experimento. Para iniciar a operação transversal é necessário encontrar o nó inicial. A API Java do *Neo4J* tem um método que pode ser usado para carregar nós da base de dados - *GraphDatabaseService.getNodeById(Long id)*. Esse método simplesmente carrega um nó a partir do seu identificador interno, o qual é representado por um número *java.util.Long number* no *Neo4j*. Após conhecer o nó inicial pode-se atravessar o grafo para encontrar os nós interessados.

Para percorrer o grafo a partir do nó inicial, utiliza-se o método *Node.getRelationship* (*Direction.OUTGOIN*) para obter todas as relações que iniciam no nó selecionado. Este método retorna um objeto *java.lang.Iterable* que contém instâncias de *Relationship*, o qual pode ser iterado usando um típico *for each* do Java. O método *Node.getRelationship* (*Direction.OUTGOIN*) irá obter todas as atividade relacionadas com o nó inicial (que

representa um experimento). Os próximos passos irão percorrer todos os relacionamentos da atividade (*USED*, *WAS_DERIVED_FROM*, *WAS_ASSOCIATED_WITH* e *WAS_GENERATED_BY*).

```
List nodes;
Node root = getNodeById(IdAccount);
for (relationship in root.getRelationship(Direction.OUTGOIN) {
  Node nodeActivity = relationship.getOtherNode(root);
  nodes.add(nodeActivity);
  for (relUsed in nodeActivity.getRelationships(USED) {
     Node nodeUsed = relUsed.getOtherNode(nodeActivity);
     nodes.add(nodeUsed);
     for(relWasDerivedFrom in nodeUsed.getRelationships(
        WAS_DERIVED_FROM)
     {
       nodes.add(relWasDerivedFrom.getOtherNode(nodeUsed);
     }
  }
  for (relWasAssociatedWith in nodeActivity.getRelationships(
     WAS_ASSOCIATED_WITH) {
    nodes.add(relWasAssociatedWith.getOtherNode(nodeActivity));
  }
  for (relWasGeneratedBy in nodeActivity.getRelationships(
     WAS_GENERATED_BY) {
    nodes.add(relWasGeneratedBy.getOtherNode(nodeActivity);
  }
  return nodes;
}
```

Listagem 4.1: Algoritmo para geração do grafo de proveniência

Uma outra alternativa para recuperar os dados é utilizar a linguagem de consulta do *Neo4J*, *Cypher*, que tem a característica de ser uma linguagem declarativa realizando buscas nos grafos através da descrição do que se precisa para recuperação dos dados. Na Listagem 4.2 é exibida uma consulta na linguagem *Cypher* que tem o objetivo de recuperar todos os dados do grafo de proveniência.

A consulta inicia identificando o nó inicial, que no caso é o número identificador do projeto. Em seguida, de forma transversal, recupera as entidades *Account* que possuem relacionamento *HAS* com o *Project* selecionado. Realizando o mesmo procedimento para recuperar as atividades.

```
start p=node(192920)
match p-[:HAS]-(account)-[:HAS]-(activity)-[r]-(collection)
where account.name="ACCOUNT_BACILLUS"
return activity, r, collection;
```

Listagem 4.2: Consulta em *Cypher* para recuperar dados de proveniência

4.1.4 Visão Geral de Um Grafo de Proveniência

Esta seção descreve, resumidamente, a tela principal do simulador para a construção de grafos de proveniência.

Ao realizar *login* no simulador, o usuário tem acesso à tela principal do sistema, conforme mostra a Figura 4.2. Esta tela pode ser dividida em três partes: exibição dos projetos, menus e visualização do conteúdo.

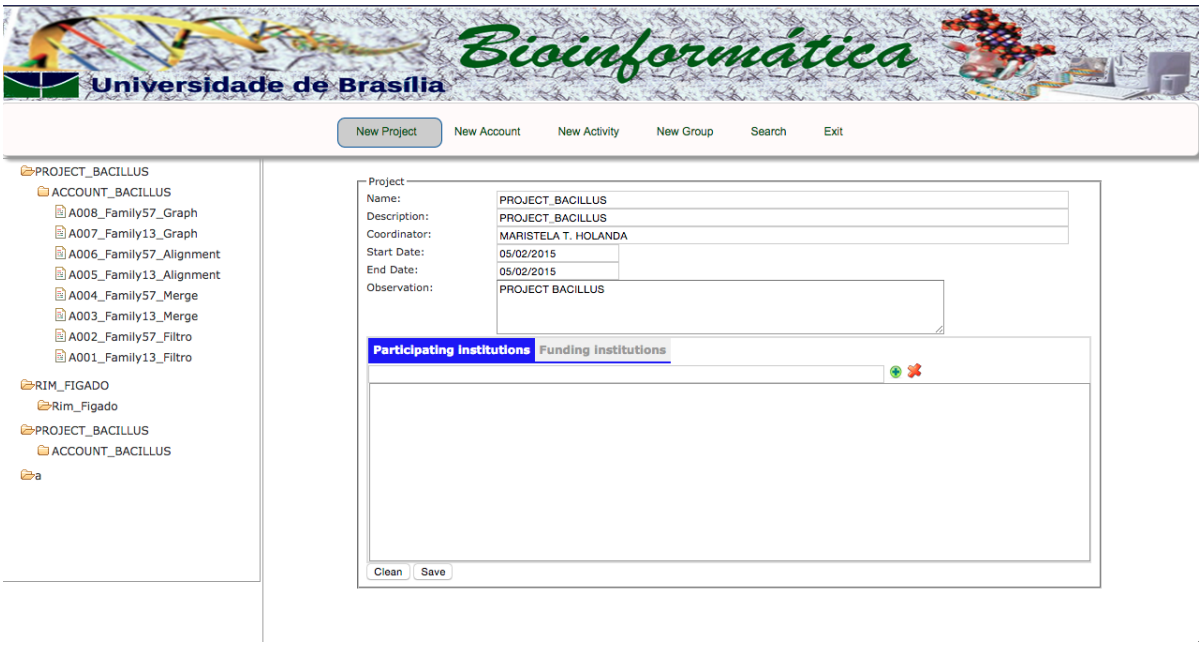

Figura 4.2: Tela principal do simulador de proveniência.

Na parte esquerda são exibidas as entidades *Project*, *Account* e *Activity* contruídas pelo usuários e agrupadas no formato de uma árvore pelo simulador. O primeiro nível da árvore é composto pela entidade *Project*, já o segundo nível contém todos os experimentos executados e associados ao projeto. E o terceiro nível contém todas as atividades que foram executadas durante o experimento. A navegação por árvore facilita a visualização das entidades associadas ao usuário, assim como uma melhor navegabilidade entre os elementos de um projeto.

Já a parte referente ao menu da Figura 4.2 são comandos que serão executados ao serem clicados. No simulador, há cinco menus que são responsáveis por exibir o formulário para cadastro das entidades, com exceção do menu *Search* que irá exibir a tela de montagem do grafo.

Já a parte central da tela contém o parâmetros necessários para a criação de cada uma das entidades. Ao se clicar em um menu da tela, a parte central irá exibir os campos

para preenchimento. Já ao selecionar qualquer elemento da árvore à esquerda, os dados do elemento selecionado são carregados na parte central para visualização ou alteração.

4.2 Estudo de Caso 1 - *Alpha-amylase*

O primeiro estudo de caso apresentado nesse trabalho trata de sequências de DNA de bactérias extremófila *Alpha-amylase* a partir do banco de dados UNIPROT (UNIPROT, 2012). Este projeto objetivou encontrar o gene que codifica uma determinada proteína no genoma do bacilo, assim como também comparar a sequência encontrada com outras disponíveis. O experimento que está sendo demonstrado nesse estudo de caso, refere-se à comparação do gene encontrado com outras sequências disponíveis. Para tal objetivo, foram realizados alinhamentos múltiplos para amostras das famílias 13 e 57 desta bactéria, e por fim foram gerados gráficos destes alinhamentos. Originalmente, esse experimento foi realizado no Laboratório de Biologia Molecular (BIOMOL) do Departamento de Biologia Celular (CEL) da Universidade de Brasília (UnB).

A Figura 4.3 trata do *workflow* do estudo de caso da *Alpha-amylase* que se inicia quando se obtém as sequências de DNA do banco de dados do UNIPROT, na qual são passadas pela fase de filtragem removendo as sequências que possam dificultar e afetar negativamente nos resultados da próxima fase. Utilizou-se *scripts* desenvolvidos em *Perl* para realizar a filtragem das *reads*. Em seguida, de acordo com a Figura 4.3 começa a fase de alinhamento, cujo objetivo é localizar dentro de um genoma de referência de um organismo próximo ao organismo estudado. O programa utilizado nessa fase foi o *clustal* (Higgins and Sharp, 1988), ferramenta utilizada para alinhamento múltiplo de sequências. Por fim, a fase de análise que é um processo de interpretação dos dados brutos gerados pelo sequenciamento com o objetivo de acrescentar informações biológicas, sendo utilizado o programa o *WebLog* para este estudo de caso.

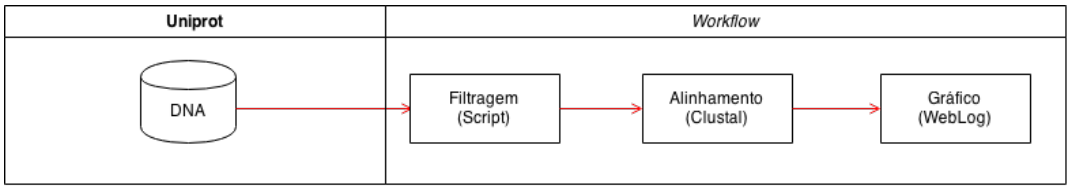

Figura 4.3: Workflow resumido do experimento com a bactéria Alpha-amylase indicando os programas utilizados.

Utilizando o protótipo implementado a partir da arquitetura proposta no Capítulo 3 para executar o experimento do *workflow* da Figura 4.3, foi gerado o grafo de proveniência das atividades executadas ao utilizar a funcionalidade de *Search* do simulador *Provenance*.

Conforme definido no modelo PROV-DM, as atividades são representadas por retângulos, enquanto que as coleções são representadas por elipses e os agentes por pentágonos com a base reta. O grafo exibido na Figura 4.4 apresenta um agente, 11 coleções e 8 atividades que estão ligadas através de 35 arestas formando o histórico do experimento, descrito a seguir.

No simulador *Provenance* esse experimento foi executado por um estudante do Departamento de Ciência da Computação (CIC) da UnB que é representado pelo agente com identificador *AG001_RODRIGO*.

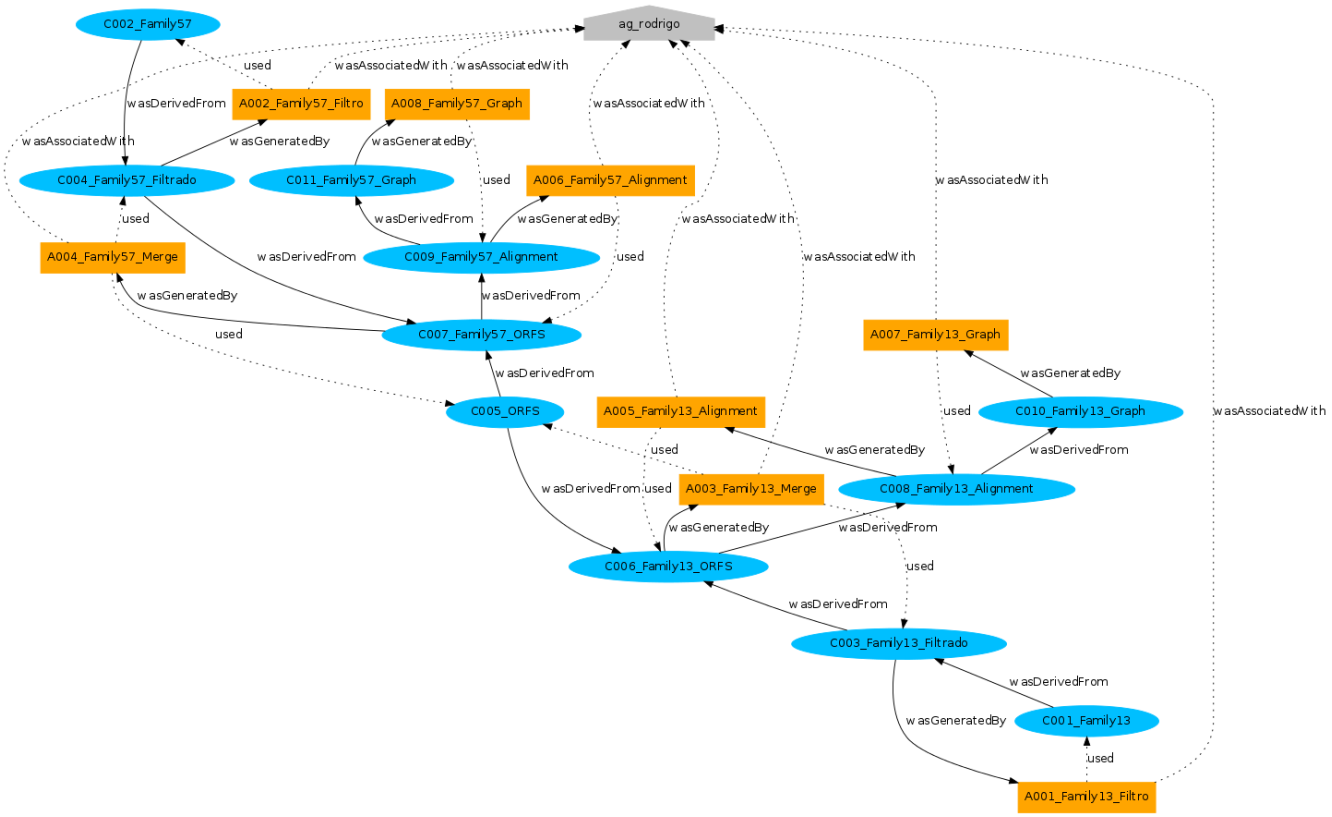

Figura 4.4: Grafo de proveniência do experimento da bactéria *Alpha-amylase*.

Na árvore esquerda da Figura 4.5 pode-se notar as entidades criadas pelo usuário para a execução do experimento pelo simulador. Observa-se que o projeto *PROJECT_ALPHA_AMYLASE* criado contém o experimento *ACCOUNT_ALPHA_AMYLASE*, que por sua vez possui um total de oito atividades. A parte central da Figura 4.5 exibe o detalhamento dos parâmetros utilizados para a criação do experimento, tais como nome, descrição, local de execução, versão, datas de início e fim do experimento.

Pelo grafo da Figura 4.4 o agente *AG001_RODRIGO* está associado a todas as atividades executadas nesse experimento. Isso pode ser visto pelas arestas entre o agente e cada atividade no grafo. Essa aresta representa a relação *WasAssociatedWith* do modelo PROV-DM. A origem é sempre a atividade e o destino é o agente.

As coleções *C001_Familia13* e *C002_Familia57* representam arquivos no formato FASTA com sequências DNA das famílias 13 e 57, respectivamente, da bactéria Alpha-amylase obtidas a partir do banco de dados do UNIPROT. A coleção *C005_ORFS* também representa um arquivo FASTA no formato de proteínas, com os *contigs* do gene encontrado.

Inicialmente foi realizado um processo de filtro, representado pela atividade *A001_Family13_Filter* que usou a coleção *C001_Family13* para a geração da coleção *C003_Family13_Filter*. A partir daí, a atividade *A003_Family13_Merge* executou um processo de junção da coleção *C003_Family13_Filter* e *C005_ORFS* gerando a coleção *C006_Family13_Filter*. A seguir a coleção *C006_Family13_ORFS* é usada pela atividade *A005_Family13_Alignment* a fim de realizar o alinhamento múltiplo das sequências dessa coleção. Esses alinhamentos são gravados em um arquivo representado pela coleção

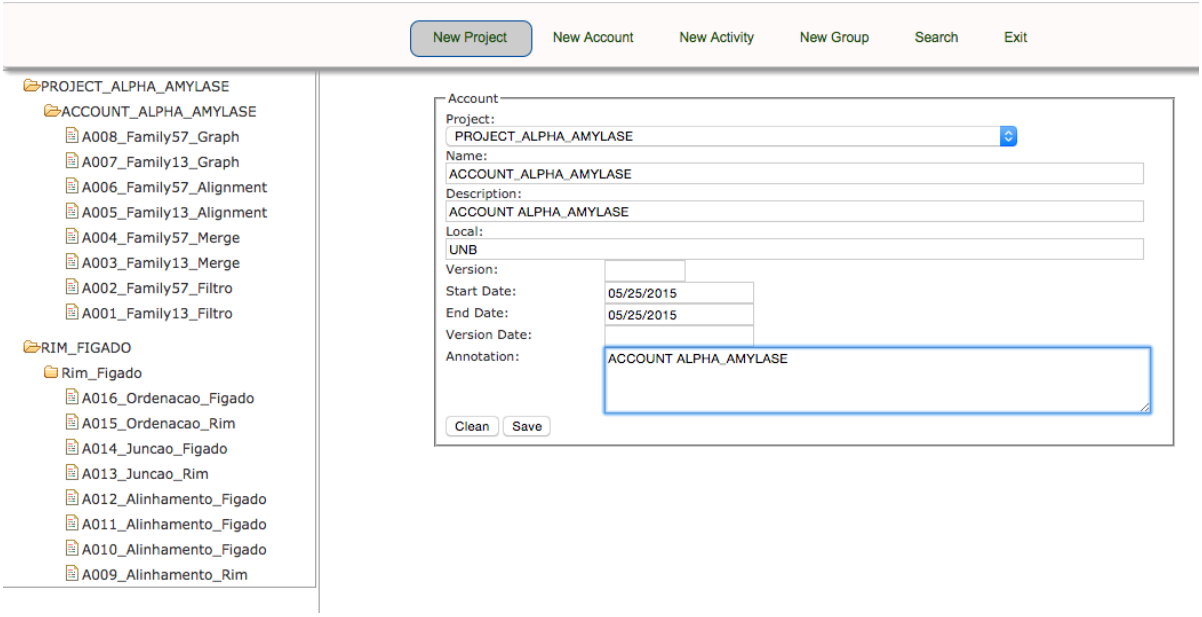

Figura 4.5: Tela de detalhes da entidade *Account* do *workflow Alpha-amylase*.

C008_ Family13 _ Alignment. E, por fim, a coleção *C010_ Family13_ Graph* é gerada pela atividade *A007_ Family13_ Graph* que usou os alinhamentos na coleção *C008_ Family13 _ Alignment*. Pode-se perceber que a mesma sequência de processo ocorre com as coleções da família 57.

No grafo da Figura 4.4, as arestas que possuem uma atividade como origem e uma coleção como destino representam a relação *Used*, e as arestas que possuem uma coleção como origem e uma atividade como destino representam a relação *WasGeneratedBy*.

Dessa forma, pode-se afirmar que o grafo da Figura 4.4 apresenta a proveniência das duas coleções *C010_ Familia13_ Grafico* e *C011_ Familia57_ Grafico*, assim como as arestas que começam nessas coleções e percorrem as demais estão demonstrando as derivações até as coleções iniciais.

Na Figura 4.6 foi selecionado o grupo *Família57* através da tela de *Search* do simulador. Essa consulta irá construir o grafo de proveniência apenas com os nós que pertencem ou se relacionam as atividades da *Família_ 57*.

Portanto, quando o objetivo é analisar apenas parte do experimento executado, a funcionalidade de agrupar os nós por certas características permite uma melhor visualização do grafo.

4.3 Estudo de Caso 2 - Rim-Fígado

O segundo estudo de caso trata do sequenciamento de amostras de células de rim e fígado para identificar a expressão diferencial de genes em comparação com tecnologias de arranjos existentes (Hoheisel, 2006). Neste contexto, o objetivo foi comparar a capacidade de identificar genes diferencialmente expressos entre duas abordagens: sequenciamento de alto desempenho e tecnologia de arranjos. O sequênciamento das amostras de *cDNA* de rim produziu 72 987 691 SRS (*Short Read Sequences*, que são fragmentos sequenciados obtidos dos sequenciadores de alto desempenho) e a amostra de fígado 72 126 823 SRS.

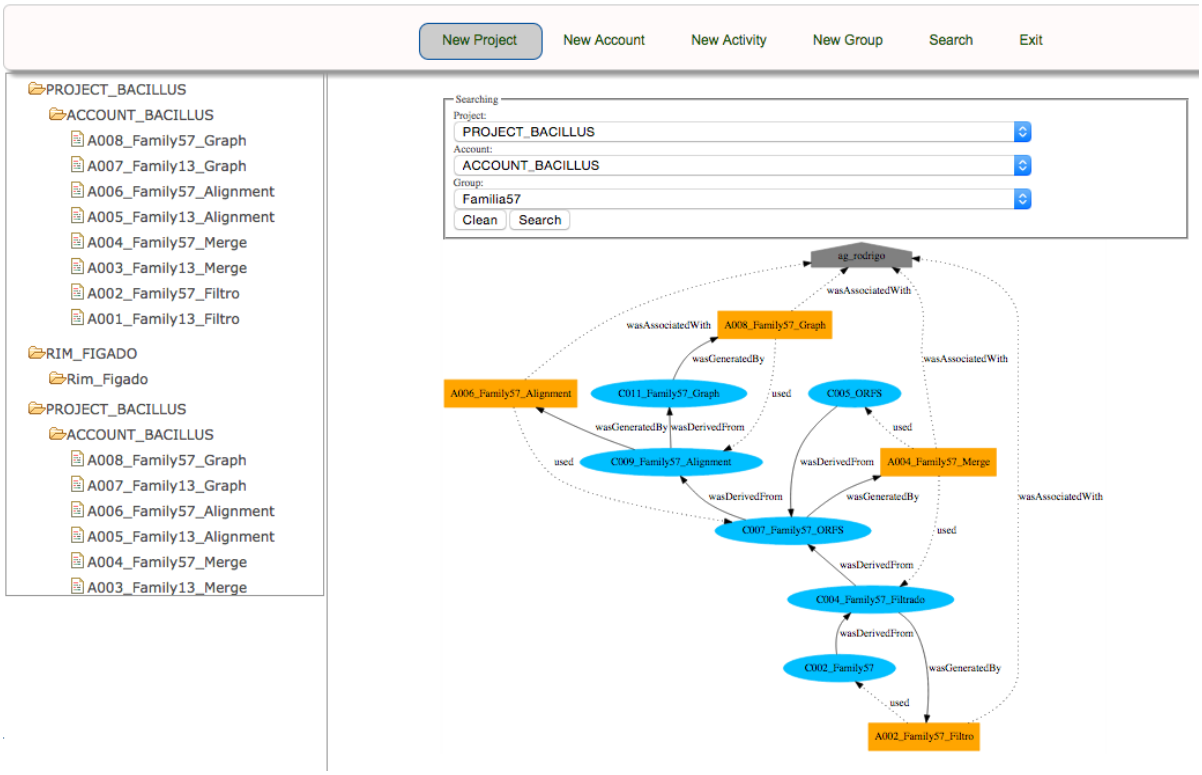

Figura 4.6: Grafo de proveniência do grupo da Família_57

O resultado do sequênciamento é um conjunto de arquivos *FASTQ* contendo SRS de 36 pares de bases de comprimento com as suas qualidades associadas a cada base.

A Figura 4.7 mostra o funcionamento do *workflow*. As SRS utilizadas no estudo de caso tem formato *FASTQ*. De forma geral, este formato é composto pelas cadeias de sequências de bases e sequências de qualidades associadas a cada base. Este tipo de arquivo armazena informações geradas pelo sequenciador *Illumina* em formato texto.

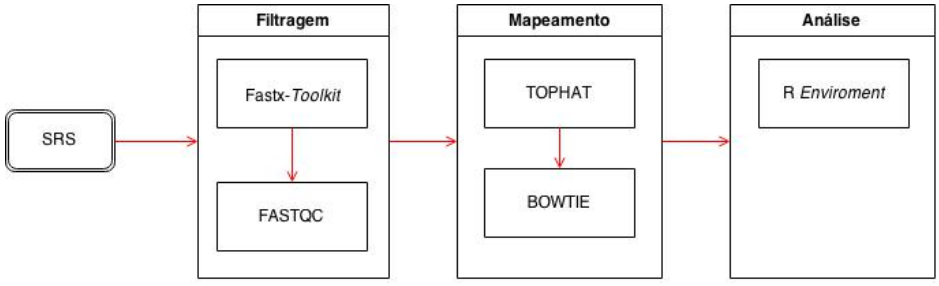

Figura 4.7: *Workflow* resumido do experimento indicando os programas utilizados.

Na fase de filtragem do *pipeline*, foram usados os pacotes FASTX-*Toolkit* e o pacote FASTQC. O FASTX-*Toolkit* (Fastx-Toolkit, 2015) é uma coleção de ferramentas que fornece pré-processamento de arquivos FASTA e FASTQ. O FASTQC (FASTQC, 2015) é uma aplicação Java que gera um relatório de controle de qualidade dos dados do sequenciamento de alto desempenho com o objetivo de detectar problemas que se originam tanto no sequenciador ou no material usado no sequenciamento de alto desempenho.

Logo, no *workflow* foi usado o FASTX-*Toolkit* para eliminar as SRS de baixa qualidade, e os pacotes FASTQC para avaliar se os resultados alcançados foram aceitáveis através de informes estatísticos. Esta fase da filtragem é de extrema importância para assegurar que a próxima fase use apenas sequências com qualidades aceitáveis.

Uma vez a fase de filtragem completada, o processo de mapeamento começa usando o programa *TopHat* (Trapnell et al., 2009), que implementa um algortimo de mapeamento de SRS eficiente para alinhamento das SRSs que vem do sequenciador de alto desempenho. Na primeira fase, são mapeadas todas as SRS no genoma de referência usando o *Bowtie* (Langmead et al., 2009). E por último foi executado o programa R (R Development Core Team, 2009) foi escolhido para implementar a análise de dados, pois se trata de um *software* livre para computação de estatísticas.

Na Figura 4.8, na árvore à esquerda pode-se ver as entidades criadas pelo usuário para a execução do experimento pelo simulador *Provenance*. Observa-se que o projeto RIM_FIGADO criado contém um experimento nomeado de Rim_Figado, que por sua vez possui um total de dezesseis atividades. A parte central da Figura 4.8 é o detalhamento dos parâmetros utilizados na atividade A001_Filtro_Rim, tais como o nome e versão do programa utilizado para execução, a data de início e fim em que a atividade foi executada, o grupo no qual a atividade pertence, a linha de comando com os seus parâmetros de configuração a ser executada e o arquivo de entrada.

A execução de uma atividade inicia-se ao se clicar no botão *Run Command* da Figura 4.8 no simulador. Neste momento uma requisição é enviada para a camada *Controller*, que por sua vez encaminha uma solicitação para o módulo de processamento que irá enviar o comando a ser executado para a fila. Ao perceber que há um comando na fila, o módulo de execução recupera essa informação, cria os diretórios necessários para a execução do comando, instancia um *Watcher* para monitoração dos diretórios e executa o comando. Finalmente, quando o processo cria ou altera qualquer arquivo, o *Watcher* é notificado, extrai as informações de proveniência e armazena no banco de dados de grafo através do módulo de armazenamento.

Após a execução do *workflow* referente ao estudo de caso do rim-fígado, utilizou-se a funcionalidde de *Search* do simulador desenvolvido e solicitou a visualização do grafo exibido através das Figura 4.10 e 4.11, que é composto por um agente, 23 coleções e 16 atividades que estão ligadas através de 84 arestas, formando o histórico do experimento, conforme descrito a seguir.

Como pode ser visto no grafo, apenas um agente (AG001_Executor) trabalhou nesse experimento, executando todas as atividades. O que pode ser visto através da relação *WasAssociatedWith* do agente com todas as atividades do grafo.

O experimento iniciou com seis arquivos contendo sequências de RNA das células de rim e fígado, cada um dos arquivos sendo representado por uma coleção (SRR002324, SRR002325, SRR002320, SRR002321, SRR002322 e SRR002323). Inicialmente cada coleção passou por um processo de filtragem a fim de garantir uma qualidade mínima das sequências, gerando novas coleções apenas com as sequências de qualidade boa. Em seguida, as coleções filtradas passaram por uma fase de alinhamento com um genoma de referência (representado pela coleção *hg19*). As coleções alinhadas são unidas para gerar uma nova coleção que sofrerá uma ordenação tendo como resultado uma coleção de sequências ordenadas.

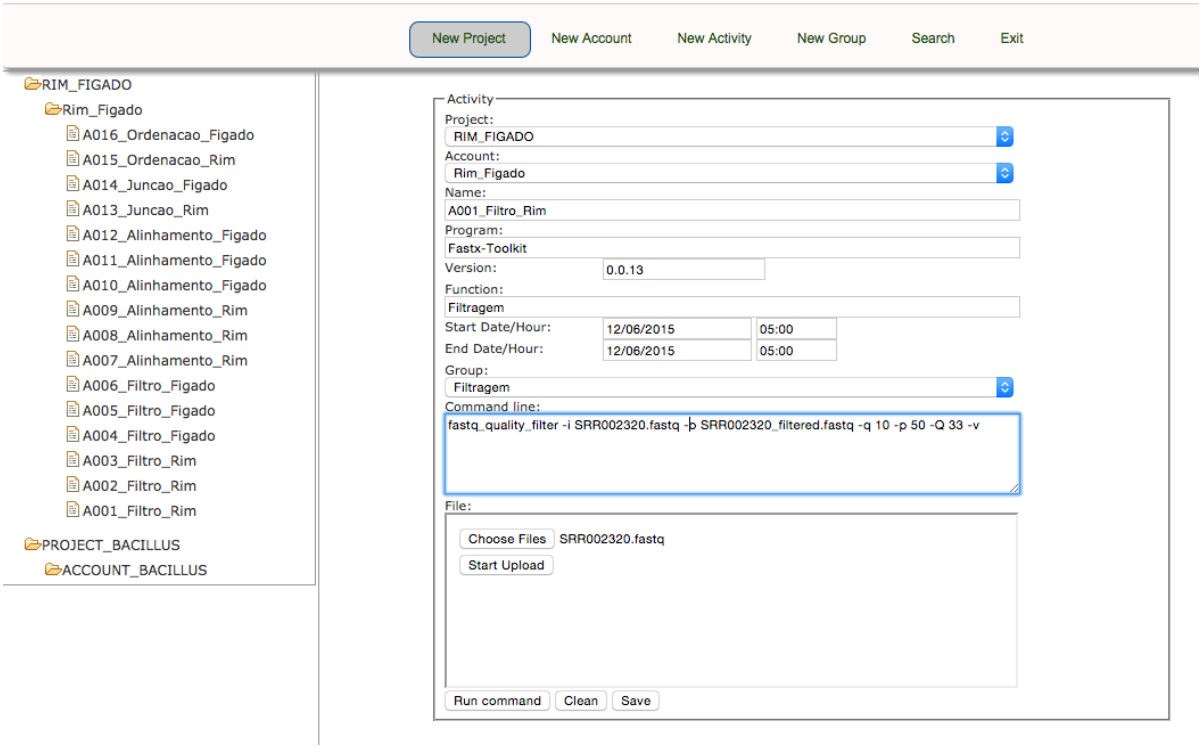

Figura 4.8: Tela da Atividade A001_Filtro_Rim.

Por exemplo, o grafo representado pelas Figuras 4.10 e 4.11 mostram que a atividade A001_Rim_Filtro usou a coleção SRR002320 para a geração da nova coleção *SRR002320_filtered*. A partir daí, a atividade A007_Alinhamento_Rim executou um processo de alinhamento das sequências (*SRR002320_filtered*) com o genoma de referência (hg19). Deste alinhamento foi gerada a coleção *SRR002320_accepted_hits*. Este mesmo processo foi realizado com as demais coleções de entrada. Em seguida, a atividade A013_Juncao_Rim foi responsável por unir as sequências alinhadas oriundas da amostra de Rim (*SRR002320_accepted_hits*, *SRR002324_accepted_hits* e *SRR002325_accepted_hits*), assim como a atividade A014_-Juncao_Figado foi responsável por unir as sequências alinhadas da amostra de Fígado (*SRR002321_accepted_hits*, *SRR002322_accepted_hits* e *SRR002323_accepted_hits*), gerando respectivamente, as coleções rim_juncao e figado_juncao. Por fim, as atividade A015_Ordenacao_Rim e A016_Ordenacao_Figado foram executadas para realizar o ordenação das coleções rim_juncao e figado_juncao, gerando as coleções rim_ordenado e figado_ordenado, respectivamente.

Portanto, pode-se ver no grafo das Figuras 4.10 e 4.11 que as arestas pontilhadas possuem uma atividade como origem e uma coleção como destino representam a relação *Used* e as arestas que possuem uma coleção como origem e uma atividade como destino representam as relações *WasGeneratedBy*. Já as arestas que possuem como origem uma coleção e como destino um agente representa a relação *WasAssociatedWith* e as arestas que tem como origem e destino uma coleção representam as derivações feitas das coleções iniciais até a geração da coleção objetivo. Dessa forma, esse grafo demonstra a proveniência das coleção rim_ordenado e figado_ordenado.

Para melhor visualização do grafo, poder-se-ia dividí-lo pelas fases do *workflow*, conforme pode ser visto na Figura 4.9, que exibe apenas os componentes da fase de filtra-

gem. Tais componentes são as coleções de entrada (SRR002320, SRR002321, SRR002322, SRR002323, SRR0023204 e SRR002325), as atividades de filtragem (A001_Filtro, A002_-Filtro, A003_Filtro, A004_Filtro, A005_Filtro e A006_Filtro), o agente associado a elas (AG001_Executor) e as coleções geradas pela execução das atividades (*SRR002320_filtered*, *SRR002321_filtered*, *SRR002322_filtered*, *SRR002323_filtered*, *SRR002324_filtered*, *SRR-002325_filtered* e *SRR002326_filtered*).

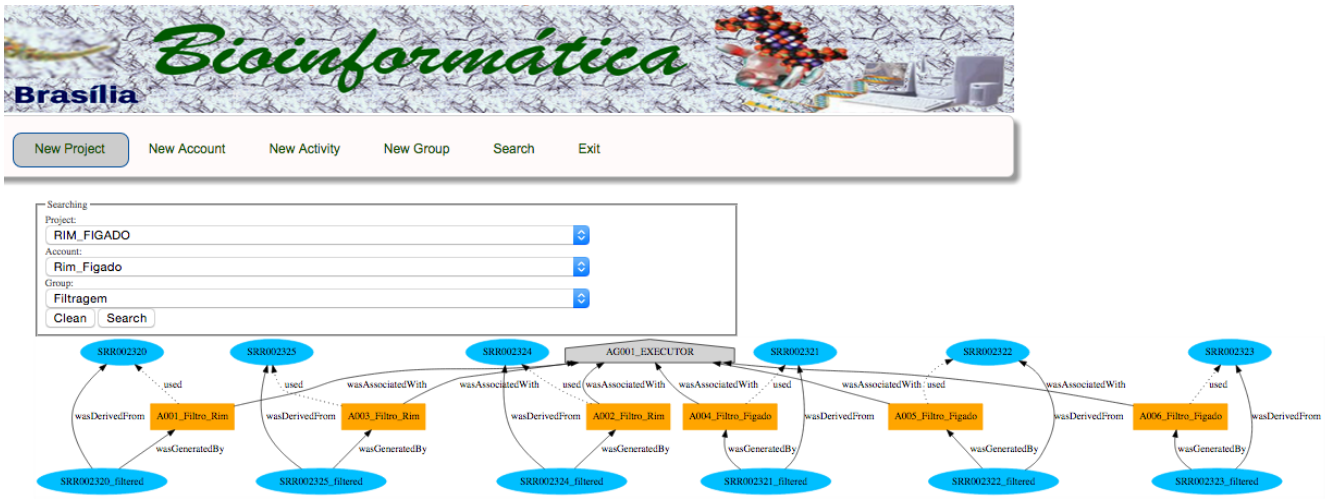

Figura 4.9: Grafo personalizado da fase de Filtragem.

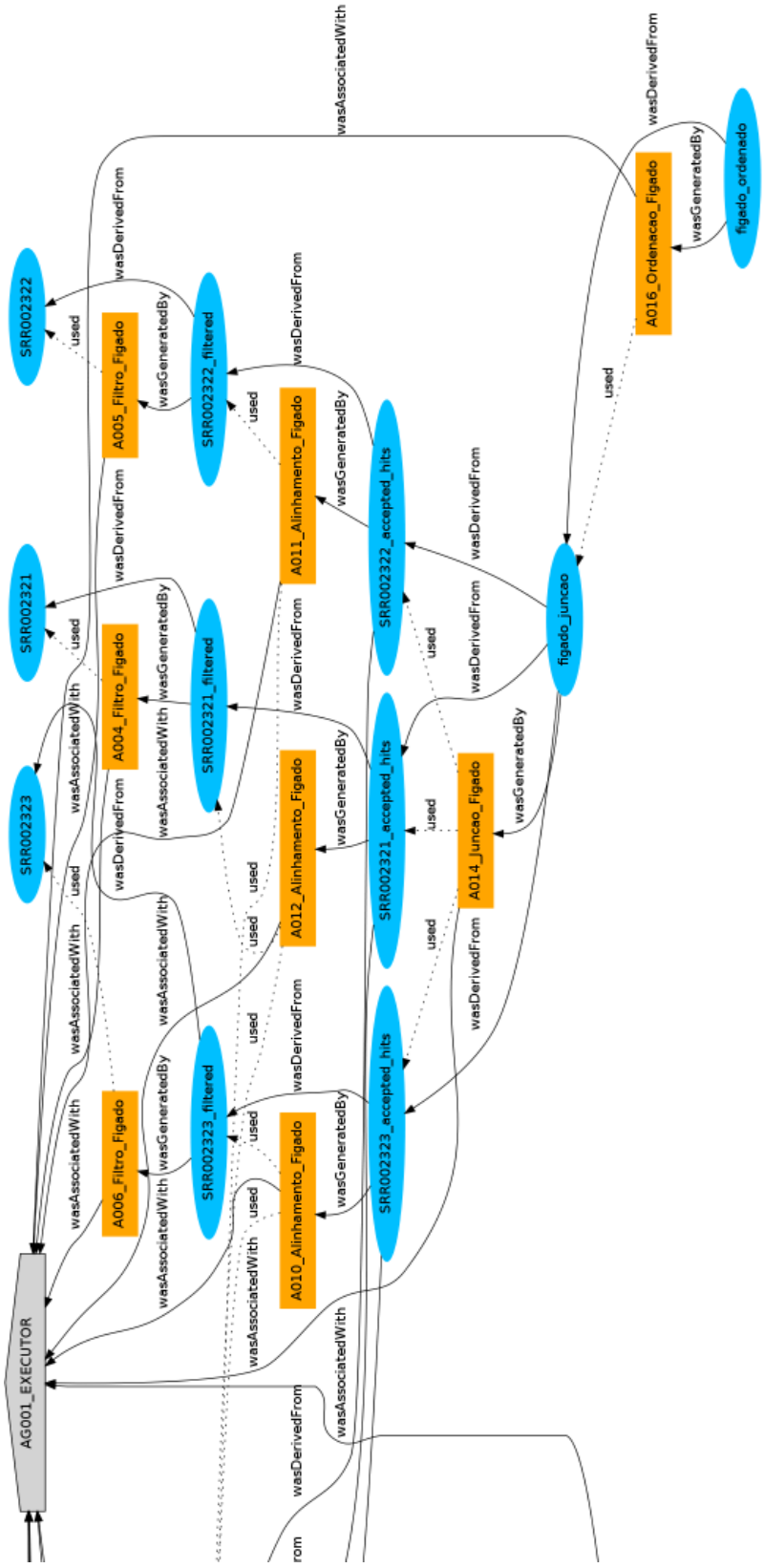

Figura 4.10: Parte I - Grafo do experimento que trata a expressão diferencial entre células de rim e fígado.

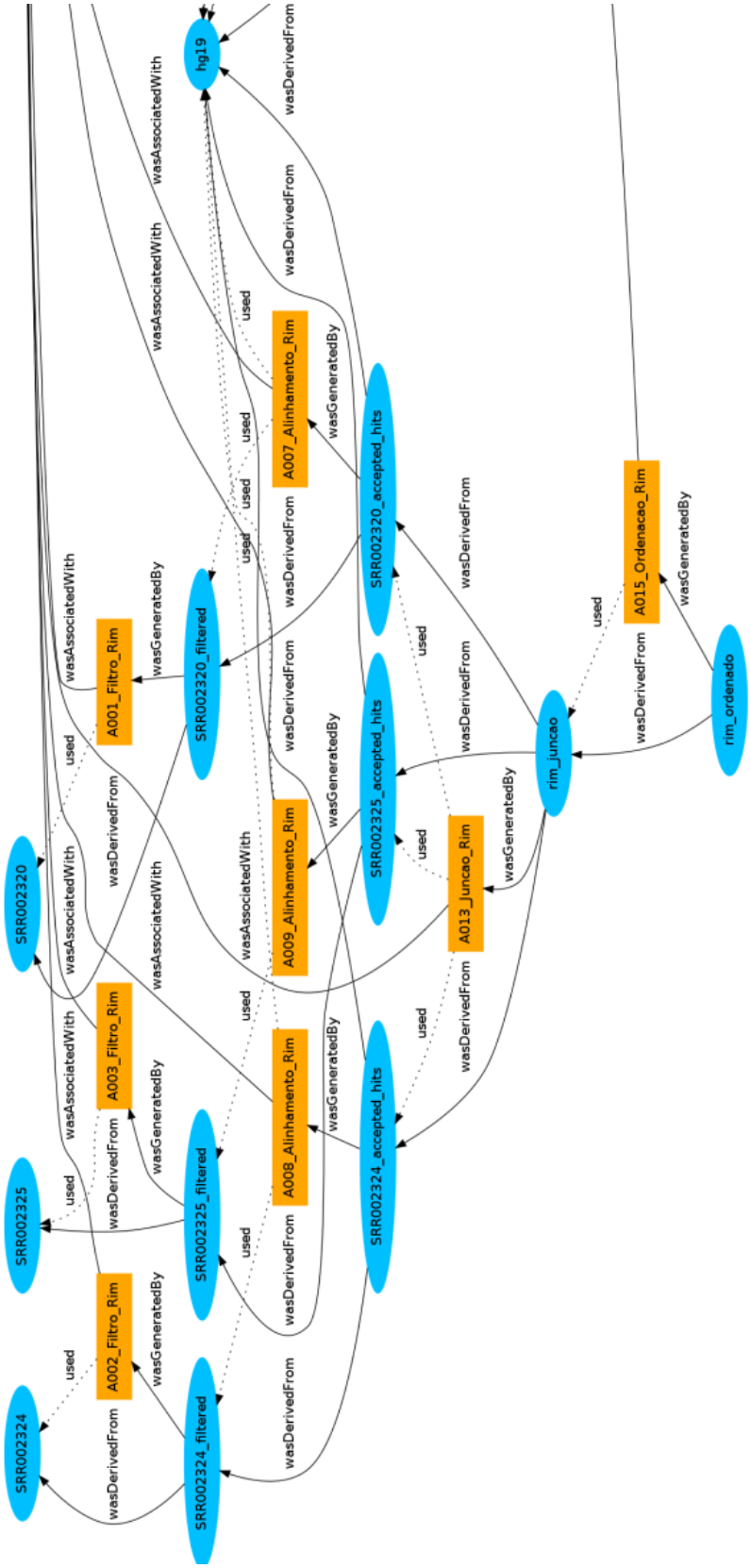

Figura 4.11: Parte II - Grafo do experimento que trata a expressão diferencial entre células de rim e fígado.

Capítulo 5

Conclusão

Neste trabalho foi apresentada uma definição de uma arquitetura de captura de proveniência de dados automatizada no contexto da bioinformática, utilizando o modelo de proveniência PROV-DM e armazenamento em bancos de dados em grafo. Para isso, foi necessário um estudo do modelo PROV-DM para representar a proveniência de dados. Uma vez que esse modelo possui os requisitos necessários para aplicação em projetos de bioinformática, entre os principais requisitos estão: capacidade de representar a proveniência de uma peça de dado, descrevendo os processos e insumos utilizados em sua geração; uma representação gráfica adequada, com diferentes símbolos para cada elemento e relações suficientes para demonstrar a proveniência de forma objetiva; símbolos para representar grandes conjuntos de dados. Além desses requisitos, outras características devem ser destacadas em relação ao modelo PROV-DM, como o extenso material disponível cobrindo diferentes aspectos da proveniência de dados e o fato de ser padrão da W3C. Sendo essas duas características importantes indicadores da capacidade de disseminação do modelo.

A proveniência pode ser definida como um grafo acíclico dirigido. Os nós no grafo podem representar objetos, tais como arquivos, programas e pessoas e esses nós podem possuir atributos. Já as arestas entre os nós indicam uma dependência entre os objetos. Além disso a representação gráfica da proveniência, na forma de um grafo direcionado, permite ao usuário rever a execução de seus experimentos de forma clara e objetiva.

Quanto a implementação da arquitetura separada por módulos irá permitir uma independência de camadas, por exemplo, a possibilidade de anexar uma outra interface sem alteração das demais camadas. Além disso facilita a manutenção e adição de recursos e reaproveitamento do código desenvolvido. A linguagem Java utilizada por ser multiplataforma permite que o simulador execute em qualquer sistema operacional.

Como resultado da implementação da arquitetura proposta foi desenvolvido o simulador Provenance para demonstrar a aplicação e captura automática do modelo PROV-DM em projetos de bioinformática. A utilização do simulador possibilita o agrupamento dos experimentos semelhantes por projetos, faciltando o gerenciamento. Por ser uma aplicação web permite o acesso remoto e não necessita de instalações em computadores locais, facilitando o acesso e compartilhamento da informação entre equipes distintas. A criação de grafos personalizados fornece ao usuário uma ferramenta capaz de demonstrar as diferentes visões de seus experimentos.

Para a definição do modelo de grafos optou-se pelo modelo de grafos de atributos, pois facilita a recuperação do dado e simplifica a diagramação porque são necessários

menos vértices. O banco de dados selecionado para implementação foi *Neo4J* devido a sua popularidade, facilidade de uso, vasta comunidade e documentação disponível na *internet*, além de permitir o armazenamento de atributos.

Quanto a captura automática realizada pelo simulador permite ao usuário focar nos detalhes e análise do experimento não precisando realizar a instalação e configuração do ambiente de execução, facilitando a re-execução e validação do experimento.

Este trabalho gerou a publicação dos artigos *Automatic capture of provenance data in genome project workflows* e *Storing provenance data of genome project workflows using graph databas*.

5.1 Trabalhos Futuros

Alguns aspectos podem ser melhorados no simulador *Provenance*, tais como o armazenamento das coleções em banco de dados que irá permitir uma maior validade dos dados, assim como agilizar a recuperação dos arquivos usados. Permitir realizar a proveniência dos dados a um nível menor, ou seja, das entidades contidas nas coleções.

Alterar a arquitetura proposta para utilizar os benefícios da computação em nuvem. Para isso, propor mais um módulo para coordenação e escalonamento das atividades executadas. Realizar a separação do banco de dados de grafo do simulador *Provenance* porque ele funciona de forma embarcada.

Uma das dificuldades encontradas durante o desenvolvimento do simulador foi a realização de *upload* de arquivos grandes, portanto um trabalho futuro seria investigar meios de melhorar o tempo e a forma de se trabalhar com grande volume de dados dos projetos de bioinformática.

Poder-se-ia criar um plano de estudo para analisar a aceitação e usabilidade do simulador *Provenance* pelos biólogos, afim de que se pudesse evoluir características e funcionalidades do simulador que facilitasse o dia-a-dia dos biólogos.

Criação de um ambiente colaborativo e social que permitesse a troca de dados do experimento entre diversos usuários e equipes distintas, permitindo a coraboração entre equipes localizadas remotamente.

Por fim, adicionar a funcionalidade de reprodução automática do experimento através do simulador *Provenance* a partir do grafo de proveniência gerado. Para isso utilizando tecnologias de provisionamento e virtualização.

Referências

Altintas, I., Barney, O., and Jaeger-Frank, E. (2006). Provenance collection support in the kepler scientific workflow system. In *Provenance and annotation of data*, pages 118–132. Springer. 1

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410. 8

Alvarez, P. (2009). Pipelines para transcritomas obtidos por sequenciadores de alto desempenho. Master's thesis, Universidade de Brasília, Brasília. iii

Angles, R. (2012a). A comparison of current graph database models. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pages 171–177. IEEE. 18

Angles, R. (2012b). A comparison of current graph database models. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pages 171–177. IEEE. 22

Angles, R. and Gutierrez, C. (2008). Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1. 17, 20

Bentley, D. R. (2006). Whole-genome re-sequencing. *Current opinion in genetics and development.*, 16(6):545–552. 1, 4

Buneman, P., Khanna, S., and Wang-Chiew, T. (2001). Why and where: A characterization of data provenance. In *Database Theory—ICDT 2001*, pages 316–330. Springer. 1

Bunge, M. (1977). Treatise on basic philosophy: Volume 3: Ontology 1: The furniture of the world. *Reidel, Boston.* 9

Cassandra (Acessado em: 05 de Abril de 2015.). Disponível em: `http://cassandra.apache.org/`. 22

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4. 22

Cormen, T. H. (2002). *Algoritmos: teoria e prática*. Elsevier. 38

Costa, F., Silva, V., de Oliveira, D., Ocaña, K., Ogasawara, E., Dias, J., and Mattoso, M. (2013). Capturing and querying workflow runtime provenance with prov: a practical approach. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 282–289. ACM. 14, 26, 28

CouchDB. (Acessado em: 05 de Abril de 2015.). Disponível em: http://couchdb.apache.org. 22

Dahm, R. (2008). Discovering dna: Friedrich miescher and the early years of nucleic acid research. *Human genetics*, 122(6):565–581. 4

Davidson, S. B. and Freire, J. (2008). Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1345–1350. ACM. 8, 9

de Oliveira, A. H. M., de Souza Martins, M., Modesto, I., de Oliveira, D., and Mattoso, M. (2012). Reprodução de experimentos científicos usando nuvens. *Anais do XVII Simpósio Brasileiro de Banco de Dados (SBBD 2012)*. 14, 26, 28

de Oliveira, D., Costa, F., Silva, V., Ocaña, K., and Mattoso, M. (2014). Debugging scientific workflows with provenance: Achievements and lessons learned. *SBBD Proceedings*, 29. 27, 28

de Paula, R., Holanda, M., Gomes, L. S., Lifschitz, S., and Walter, M. E. M. (2013). Provenance in bioinformatics workflows. *BMC bioinformatics*, 14(Suppl 11):S6. 9, 14, 15, 25, 28, 29

Dominguez-Sal, D., Martinez-Bazan, N., Muntes-Mulero, V., Baleta, P., and Larriba-Pey, J. L. (2011). A discussion on the design of graph database benchmarks. In *Performance Evaluation, Measurement and Characterization of Complex Systems*, pages 25–40. Springer. 17, 18, 21

Dynamodb (Acessado em: 10 de Abril de 2015.). Disponível em: http://aws.amazon.com/dynamodb/. 22

Ellson, J., Gansner, E. R., Koutsofios, E., North, S. C., and Woodhull, G. (2004). Graphviz and dynagraph—static and dynamic graph drawing tools. In *Graph drawing software*, pages 127–148. Springer. 43

Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional. 37

FASTQC (Acessado em: 25 de Maio de 2015.). Disponível em: www.bioinformatics.bbsrc.ac.uk/projects/fastqc/Help. 49

Fastx-Toolkit (Acessado em: 25 de Maio de 2015.). Disponível em: http://hannonlab.cshl.edu/fastx_toolkit/index.html. 6, 49

FlockDB (Acessado em: 06 de Abril de 2015.). Disponível em: https://github.com/twitter/flockdb. 22

Frishman, D. and Valencia, A. (2009). *Modern genome annotation: the BioSapiens Network*. Springer Science & Business Media. iii

Gonçalves, J., de Oliveira, D., Ocaña, K., Ogasawara, E., Dias, J., and Mattoso, M. (2013). Performance analysis of data filtering in scientific workflows. *Journal of Information and Data Management*, 4(1):17. 14, 26, 28

HamsterDB (Acessado em: 10 de Abril de 2015.). Disponível em: http://hamsterdb.com/. 22

Hartig, O. and Zhao, J. (2010). Publishing and consuming provenance metadata on the web of linked data. In *Provenance and annotation of data and processes*, pages 78–90. Springer. 10

HBase (Acessado em: 05 de Abril de 2015.). Disponível em: https://hbase.apache.org/. 22

Higgins, D. G. and Sharp, P. M. (1988). Clustal: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1):237–244. 8, 46

Hoheisel, J. D. (2006). Microarray technology: beyond transcript profiling and genotype analysis. *Nature reviews genetics*, 7(3):200–210. 48

Huacarpuma, R., Holanda, M., Lifschitz, S., and Walter, M. (2011). A database schema for high-throughput sequencing transcriptone pipelines. In *proceedings of IADIS International Conference on Applied Computing*, pages 187–194. iii

InfiniteGraph (Acessado em: 06 de Abril de 2015.). Disponível em: http://www.objectivity.com/infinitegraph. 22

Jarrard, R. D. (2001). Disponível em: http://www.iibhg.ukim.edu.mk/obrazovanie/sm_all.pdf. 1

jQuery (Acessado em 10 de Junho de 2015.). Disponível em: https://jquery.com. 41, 42, 43

Korolev, V. and Joshi, A. (2014). Prob: A tool for tracking provenance and reproducibility of big data experiments. *Reproduce'14. HPCA 2014*. 27, 28

Langmead, B., Trapnell, C., Pop, M., Salzberg, S. L., et al. (2009). Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biol*, 10(3):R25. 7, 50

Lecheta, R. R. (2014). *AWS para Desenvolvedores Aprenda a instalar aplicações na nuvem da Amazon AWS*. Novatec Editora LTDA. ISBN 978-85-7522-393-2. 42

Levene, P. and London, E. (1929). The structure of thymonucleic acid. *The Journal of Biological Chemistry*, 83:793–802. 4

Mardis, E. R. (2008). Next-generation dna sequencing methods. *Annu. Rev. Genomics Hum. Genet.*, 9:387–402. 4

MongoDB (Acessado em: 05 de Abril de 2015.). Disponível em: https://www.mongodb.org/. 22

Moreau, L., Freire, J., Futrelle, J., Myers, J., and Paulson, P. (2009). Governance of the open provenance model. *URL http://twiki. ipaw. info/pub/OPM/WebHome/governance. pdf*. 10

Moreau, L., Ludäscher, B., Altintas, I., Barga, R. S., Bowers, S., Callahan, S., Chin, G., Clifford, B., Cohen, S., Cohen-Boulakia, S., et al. (2008). Special issue: The first provenance challenge. *Concurrency and computation: practice and experience*, 20(5):409–418. 2

Neo4J (Acessado em: 06 de Abril de 2015.). Disponível em: http://www.neo4j.org/learn/neo4j. 22

Oliveira, W., Neves, V. C., Ocaña, K., Murta, L., de Oliveira, D., and Braganholo, V. (2014). Captura e consulta a dados de proveniência retrospectiva implícita intra-atividade. *SBBD Proceedings*, 29. 14, 27, 28

Olson, M. A., Bostic, K., and Seltzer, M. I. (1999). Berkeley db. In *USENIX Annual Technical Conference, FREENIX Track*, pages 183–191. 22

OPM (Acessado em 07 de Julho de 2015.). Opm - open provenance model. Disponível em: http://twiki.ipaw.info/bin/view/Challenge/OPM. 1

Oracle (Acessado em: 23 de Março de 2015.). Disponível em: http://docs.oracle.com/javase/tutorial/essential/io/notification.html/. 39

Oracle9i (Acessado em 13 de Junho de 2015.). Disponível em: http://docs.oracle.com/cd/B10018_07/index.htm. 42

Padhy, R. P., Patra, M. R., and Satapathy, S. C. (2011). Rdbms to nosql: reviewing some next-generation non-relational databases. *International Journal of Advanced Engineering Science and Technologies*, 11(1):15–30. 22

Partner, J., Vukotic, A., and Watt, N. (2013). *Neo4j in Action*. O'Reilly Media. 40

Paulino, C., Oliveira, D., Cruz, S., Campos, M. L. M., and Mattoso, M. (2010). Captura de metadados de proveniência para workflows científicos em nuvens computacionais. *Anais do XXV Simpósio Brasileiro de Banco de Dados*. 14, 25, 28

Pinheiro, R., Aires, B., Araujo, A. F., Holanda, M., Walter, M. E., and Lifschitz, S. (2014). Storing provenance data of genome project workflows using graph database. In *Bioinformatics and Biomedicine (BIBM), 2014 IEEE International Conference on*, pages 16–22. IEEE. 25

Pinheiro, R., Holanda, M., Araujo, A., Walter, M., and Lifschitz, S. (2013). Automatic capture of provenance data in genome project workflows. In *Bioinformatics and Biomedicine (BIBM) IEEE International Conference on*, pages 15–20. 23, 26, 28

R Development Core Team (2009). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0. 8, 50

Ram, S. and Liu, J. (2007). Understanding the semantics of data provenance to support active conceptual modeling. In *Active conceptual modeling of learning*, pages 17–29. Springer. 9

RavenDB (Acessado em: 06 de Abril de 2015.). Disponível em: https://ravendb.net/. 22

Redis (Acessado em: 19 de Abril de 2015.). Disponível em: http://redis.io/. 22

Riak (Acessado em: 01 de Abril de 2014.). Disponível em: http://basho.com/riak/. 22

Roberts, A., Pimentel, H., Trapnell, C., and Pachter, L. (2011). Identification of novel transcripts in annotated genomes using rna-seq. *Bioinformatics*, 27(17):2325–2329. 8

Rothberg, J. M. and Leamon, J. H. (2008). The development and impact of 454 sequencing. *Nature biotechnology*, 26(10):1117–1124. 1, 4

Sadalage, J. P. and Fowler, M. (2013). Nosql essencial: Um guia conciso para o mundo emergente da persistência poliglota. ed. *Novatec Editora LTDA*. 22

Sahoo, S. S. and Sheth, A. P. (2009). Provenir ontology: Towards a framework for escience provenance management. *Kno.e.sis Publications*. 10

Sanger, F., Coulson, A., Friedmann, T., Air, G., Barrell, B., Brown, N., Fiddes, J., Hutchison, C., Slocombe, P., and Smith, M. (1978). The nucleotide sequence of bacteriophage φx174. *Journal of molecular biology*, 125(2):225–246. 4

Schuster, S. C. (2008). Next-generation sequencing transforms today's biology. *Nature methods*, 5(1):16–18. 1

Setubal, J. C. and Meidanis, J. a. S.-M. (1997). *Introduction to computational molecular biology*. PWS Pub. 4

Stein, L. (2001). Genome annotation: from sequence to biology. *Nature reviews genetics*, 2(7):493–503. iii

Tan, W. C. (2004). Research problems in data provenance. *IEEE Data Eng. Bull.*, 27(4):45–52. 8

Tomcat, A. (Acessado em 13 de Junho de 2015.). Disponível em: https://tomcat.apache.org/index.html. 42

Trapnell, C., Pachter, L., and Salzberg, S. L. (2009). Tophat: discovering splice junctions with rna-seq. *Bioinformatics*, 25(9):1105–1111. 8, 50

UNIPROT (2012). Disponível em: http://www.uniprot.org. 46

Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., and Wilkins, D. (2010). A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th annual Southeast regional conference*, page 42. ACM. 22, 23

Viz (Acessado em 10 de Junho de 2015.). Disponível em: https://github.com/mdaines/viz.js/. 42

VizJS (2015). Repository viz.js. iii

Voldemort, P. (Acessado em: 10 de Abril de 2015.). Disponível em: http://www.project-voldemort.com/voldemort/. 22

Vraptor (Acessado em: 04 de Março de 2015.). Disponível em: http://www.vraptor.org/pt/. 41, 42

W3C (Acessado em 06 de Abril de 2015.). Disponível em: www.w3.org/TR/prov-dm/. vi, 1, 10, 11, 12, 14

Watson, J. D., Crick, F. H., et al. (1953). Molecular structure of nucleic acids. *Nature*, 171(4356):737–738. vi, 4, 5

WebSphere, I. (Acessado em 13 de Junho de 2015.). Disponível em: http://www-03.ibm.com/software/products/pt/appserv-was. 42

West, D. B. et al. (2001). *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River. 15

Wildfly (Acessado em 13 de Junho de 2015.). Disponível em: http://wildfly.org/. 42

Woodman, S., Hiden, H., Watson, P., and Missier, P. (2011). Achieving reproducibility by combining provenance with service and workflow versioning. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*, pages 127–136. ACM. 27, 28