UNIVERSIDADE FEDERAL DE SÃO CARLOS
PROGRAMA DE PÓS-GRADUAÇÃO EM BIOTECNOLOGIA

DESENVOLVIMENTO DE UM *PIPELINE* PARA ANÁLISE GENÔMICA E TRANSCRIPTÔMICA COM BASE EM *WEB SERVICES*

HENRIQUE VELLOSO FERREIRA MELO

SÃO CARLOS
2009

HENRIQUE VELLOSO FERREIRA MELO

DESENVOLVIMENTO DE UM *PIPELINE* PARA ANÁLISE GENÔMICA E TRANSCRIPTÔMICA COM BASE EM *WEB SERVICES*

Dissertação submetida ao Programa de Pós-Graduação em Biotecnologia da Universidade Federal de São Carlos, como parte dos requisitos para obtenção do título de Mestre em Biotecnologia.

Orientadores:

Prof. Dr. Mauro Biajiz

Pror. Dr. Flávio Henrique da Silva

SÃO CARLOS

2009

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

M528dp

Melo, Henrique Velloso Ferreira.
Desenvolvimento de um *pipeline* para análise genômica e transcriptômica com base em *Web services* / Henrique Velloso Ferreira Melo. -- São Carlos : UFSCar, 2009.
105 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2009.

1. Bioinformática. 2. Genômica. 3. Sequenciamento. I. Título.

CDD: 570.285 (20ª)

Henrique Velloso Ferreira Melo

Dissertação de Mestrado submetida a Coordenação do Programa de Pós-Graduação em Biotecnologia, da Universidade Federal de São Carlos, como requisito parcial para a obtenção do título de Mestre em Biotecnologia

Aprovado em: 04/09/2009

BANCA EXAMINADORA

Prof. Dr. Flavio Henrique Silva (Orientador)
universidade Federal de São Carlos – DGE/UFSCar

Prof. Dr. José Miguel Ortega
UFMG

Prof. Dr. Lucas Bleicher
Instituto de Física de São Carlos – USP/São Carlos

OS QUE PASSAM POR NÓS

Aqueles que passam por nós,
não vão sós,
não nos deixam sós.

Deixam um pouco de si,
Levam um pouco de nós.

Antoine de Saint-Exupéry

Dedico este trabalho ao meu orientador e amigo Mauro Biajiz. Mauro, este foi o maior resultado do seu trabalho: dezenas de pessoas que se espelharam no seu exemplo de coragem, amor e dignidade. Em cada um de nós, você viverá para sempre.
Esta vitória é sua.

AGRADECIMENTOS

A meu pai Cláudio e meu irmão Marcelo, por terem me apoiado em todos os momentos com amor e carinho incondicionais.

A meus avós Laura, Eneida, José Melo - exemplos de dignidade e caráter - e a todos os demais membros de minha família pelo afeto e apoio.

À Tiça, pelo apoio e carinho maternal e à Carol, minha eterna irmãzinha.

Ao professor José Miguel Ortega e demais amigos (e futuros colegas) do Laboratório de Biodados da UFMG. Em meio a tantas adversidades, a realização deste trabalho não seria possível sem o auxílio deles.

A Daniel Cugler, Tiago Gaspar, Erick Melo, Cristiane Yaguinuma, Juliano Blanco e todos os demais amigos do Lince e do DC-UFSCar: pela amizade, apoio e por compartilharem comigo os grandes desafios dessa jornada.

A Andrews Sousa, João Márcio Lima, Claude Arrabal, Andréia Cardoso e demais amigos de São Carlos pela amizade, apoio e companheirismo desde o início da caminhada.

Aos professores César Teixeira, Maria da Graça Pimentel e Antônio Francisco do Prado, pela amizade, compreensão e apoio.

A meu orientador Flávio Henrique Silva, pelo apoio e compreensão e à Fernando Fonseca P. de Paula, pelo apoio técnico e fornecimento dos dados de sequenciamento.

Aos colegas, professores e funcionários do Programa de Pós-Graduação em Biotecnologia da UFSCar, pela cooperação e companheirismo.

À FAPESP, da qual fui bolsista no período de realização deste trabalho (processo 2009/00673-0). O projeto que participei como bolsista contribuiu de forma decisiva para a realização deste trabalho

A todos que estiveram envolvidos direta ou indiretamente na realização deste trabalho, e também àqueles que fazem da ciência um meio de melhorar o mundo e o Ser Humano.

Agradecimentos especiais a duas pessoas que estão agora em planos distantes, mas sempre perto no coração:

- Ao meu orientador Mauro Biajiz, pelo exemplo de um guerreiro que acreditou e lutou até o último segundo.

- À minha mãe, meu maior exemplo de vida; pois se hoje sou alguém com dignidade, honestidade e caráter, ela foi quem plantou as sementes.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Sistemas de *pipeline* para análise de genomas e transcriptomas têm o objetivo de criar pontes de comunicação entre as diferentes ferramentas no intuito de reduzir os esforços do pesquisador no processo de análise. A maioria dos *pipelines* descritos na literatura carece de funcionalidades importantes para o desenvolvimento de projetos de sequenciamento. Entre elas, a capacidade de acompanhar e gerar resultados parciais das análises ao longo do desenvolvimento do projeto; a presença de um ambiente colaborativo onde os diferentes laboratórios envolvidos possam contribuir com novos dados e cromatogramas; a possibilidade da configuração dos parâmetros da análise; o suporte a múltiplos *pipelines* com diferentes configurações; e o suporte à inclusão de novos programas e módulos. Neste trabalho, foi desenvolvido um protótipo que supre essas deficiências. O progresso dos projetos é acompanhado ao longo de todo o seu desenvolvimento. Para isso, recebe dados brutos de cromatogramas, realiza análises dos dados parciais e emite relatórios com os resultados. A comunicação com o servidor de processamento é realizada via *Web service*, oferecendo uma interface na linguagem universal XML que permite que aplicações cliente em plataformas heterogêneas submetam dados e realizem operações e consultas. Os *pipelines* são configurados através de arquivos XML em formato específico, no qual o pesquisador define os programas a parâmetros a utilizar. O protótipo dá suporte a múltiplos *pipelines* com execução simultânea em um mesmo projeto. A execução dos *pipelines* é realizada em paralelo por meio de um *pool de threads*, o que aumenta a eficiência dividindo a carga de processamento em servidores com mais de um núcleo. Uma aplicação cliente foi desenvolvida na plataforma colaborativa, permitindo que os diferentes grupos de pesquisa envolvidos no sequenciamento criem pipelines, visualizem resultados e troquem informações sobre o andamento do projeto. Outro diferencial do protótipo desenvolvido é a extensibilidade. Cada etapa do pipeline é encapsulada em um módulo. Novos módulos podem ser facilmente inseridos sem a necessidade de recompilação de todo o sistema, bastando para isso que o mesmo implemente uma interface específica. A inserção no sistema é realizada declarativamente em arquivos XML. Um estudo de caso foi realizado com a submissão de cromatogramas a partir do sequenciamento de ESTs (Expressed Sequence Tags) de *Sphenophorus Levis*. Um *pipeline* foi configurado para o estudo, e sua execução mostrou que o sistema é eficiente e apresenta bons resultados.

Palavras-chave: Análise Genômica, Análise Transcriptômica, *Pipeline, Web service*

ABSTRACT

Pipeline systems for genomic and transcriptomic analysis aim to create communication bridges among the existing analysis tools, therefore reducing researchers efforts. Most of the pipelines found in the literature lack important features which would be useful to the development of genome or transcriptome sequencing projects. Among them, the capacity of tracking the project results along its development, including the generation of partial reports; the presence of a collaborative environment where the involved laboratories can contribute with new data and chromatograms; the possibility to configure analysis parameters; multiple pipeline support and the possibility to include new tools and modules. In this work, a pipeline prototype was developed to overcome these shortcomings. Sequencing projects progresses are tracked along all over their developments. Chromatograms are progressively received along the development of the project and partial reports over newly received data are generated. The communication with the processing server is done via Web service, which offers a universal language interface, allowing client applications in heterogeneous platforms to submit data and execute operations and queries. Pipelines are configured in XML documents written in a predefined format, through which the researchers choose the tools and parameters to be used. The prototype offers support to multiple pipelines executed simultaneously in the same project. Pipelines are executed in parallel by the means of thread pools, what increases efficiency by distributing the workload in multiprocessed systems. Another feature of the prototype is the extensibility as each pipeline step is wrapped in a module. New modules can be easily inserted in the system through the implementation of a programming interface, therefore without the needing of recompilation. Module insertions are done in a declarative way through XML documents. A client application was also developed in the collaborative platform Sakai, allowing different research groups involved in a sequencing project to create pipelines, view results and exchange information on the project current status. To evaluate the efficiency of the prototype, a case study was carried out. Sequences generated from sequencing of *Sphenophorus levis* transcriptome were submitted and a pipeline was configured to analyze the data. The case study has pointed out that the prototype is efficient and produces good results.

Keywords: Genomic Analysis, Transcriptomic Analysis, Pipeline, Web Service

1. INTRODUÇÃO

O volume de dados genômicos disponíveis em bancos de dados públicos tem crescido exponencialmente. Esse fato demonstra o enorme interesse da comunidade científica pelo sequenciamento genômico, o qual tem sólido fundamento. O estudo do DNA de diferentes espécies tem levado à descoberta de genes com enorme importância para as mais diversas áreas.

Antibióticos, produtos de uso industrial, agentes despoluidores são alguns exemplos do potencial escondido nas cadeias de DNA.

Ao mesmo tempo, junto ao crescimento no volume de dados, aumenta a dificuldade de se lidar com eles. É necessário decodificar, entender e integrar toda essa informação. Para isso, os cientistas contam com a ajuda fundamental dos sistemas computacionais.

Inúmeras ferramentas computacionais têm sido criadas para lidar com o grande volume de informações genômicas geradas nas últimas décadas. Essas ferramentas têm tido sucesso em seus objetivos, porém, a dificuldade de comunicação entre elas é grande, uma vez que as mesmas surgiram de forma descentralizada.

Essa dificuldade de comunicação entre os programas é um desafio aos biólogos, já que são necessários processos de conversão de dados. Se realizada dessa forma, a análise genômica se transforma em um processo demorado, complexo e entediante.

Sistemas de *pipeline* para análise genômica têm o objetivo de contornar esses obstáculos, criando pontes de comunicação entre as diferentes ferramentas no intuito de reduzir os esforços do pesquisador no processo de análise genômica.

1.1 Motivação

Atualmente existem na literatura diversos sistemas de *pipeline* para análise genômica. Na seção 4 deste trabalho é realizada uma revisão de alguns dos *pipelines* existentes e suas principais funcionalidades. Muitos deles executam com sucesso a análise de dados genômicos, produzindo resultados satisfatórios. Porém, algumas características importantes, as quais serão citadas adiante, não podem ser encontradas na maioria desses sistemas.

O sequenciamento de genes é um processo gradual. Os projetos geralmente levam meses ou anos para a conclusão e normalmente contam com a participação de muitos laboratórios, cada qual sequenciando uma parte do DNA (ou cDNA) da espécie-alvo (no entanto, essa é uma realidade que está mudando com a introdução do pirosequenciamento).

Para se adequar a projetos dessa natureza, é muito importante que a análise computacional dos dados seja um processo gradativo. O *pipeline* deve ser capaz de acompanhar o projeto ao longo de todo o seu progresso, recebendo dados brutos de fluorescência gerados no seqüenciador periodicamente e gerando relatórios dos dados recebidos. Para isso, o sistema deve gerenciar informações do projeto, laboratórios participantes e submissões realizadas. A maioria dos *pipelines* estudados não apresenta essas funcionalidades. As análises nesses sistemas são pontuais, isto é, todas as sequências são processadas de uma só vez, o que é incompatível com a natureza gradativa dos projetos genômicos.

Outro ponto de deficiência nos sistemas estudados é a falta de opções de configuração do *pipeline.* Geralmente esses sistemas executam um único *pipeline* que já é previamente parametrizado. No entanto, testar diferentes parâmetros de análise pode ser uma boa estratégia no sentido de se otimizar a análise para se obter os melhores resultados. Além disso, pode acontecer de laboratórios envolvidos no sequenciamento discordarem quanto aos parâmetros a utilizar na análise. Um deles pode considerar, por exemplo, que a identidade mínima ideal para alinhamento entre sequências incluídas em um *cluster* é de 95%, enquanto outro considera ser 98%. Na maioria dos projetos atuais, teria que se chegar a um consenso quanto a essas divergências. Por isso, seria interessante que uma ferramenta de análise fosse capaz de permitir a criação de múltiplos *pipelines*, cada qual permitindo a configuração os parâmetros a utilizar. Seria possível, desse modo, a comparação e a discussão dos dados obtidos nas diferentes análises.

Além das funcionalidades citadas, é interessante a integração de sistemas de análise genômica a ambientes colaborativos *online* para compartilhamento dos dados do projeto. Esses ambientes devem permitir que os integrantes do projeto criem *pipelines* personalizados, compartilhem seus resultados, e ao mesmo tempo, comuniquem-se e troquem informações com outros membros.

1.2 Objetivos

O objetivo desse trabalho, com base nas motivações descritas na seção anterior, é a construção de um protótipo de *pipeline* para análise genômica e transcriptômica incluindo as seguintes funcionalidades:

- Gerenciamento de dados do projeto, laboratórios, bibliotecas genômicas e submissões de cromatogramas.
- Suporte a múltiplos projetos.

- Suporte à personalização de parâmetros de *pipeline*.
- Suporte a múltiplos *pipelines.*
- Geração de relatórios parciais ao longo do progresso do projeto.

Também é objetivo deste projeto a implementação de um sistema *Web* disponibilizando um ambiente colaborativo para o desenvolvimento de projetos de sequenciamento. Esse sistema, além de administrar dados do projeto e disponibilizar ferramentas para a troca de informações entre os membros do projeto, deve permitir que os usuários criem *pipelines* personalizados e compartilhem seus resultados.

1.3 Organização

Este documento está organizado da forma descrita a seguir.

No capítulo 2 são definidos termos importantes para o entendimento do restante do trabalho.

No capítulo 3 são apresentados o conceito e a descrição funcionamento de *pipelines* de análise genômica.

No capítulo 4 é apresentada a revisão bibliográfica. Nesse capítulo, os principais sistemas de análise genômica existentes na atualidade são descritos. São também apresentadas vantagens e deficiências de cada um.

Os capítulos 5 e 6 apresentam o protótipo de *pipeline* de análise genômica proposto neste trabalho. São discutidos também os motivos, as vantagens e desvantagens da abordagem proposta.

O capítulo 7 dá detalhes técnicos de implementação do protótipo.

O capítulo 8 descreve um estudo de caso com o protótipo em sequências reais. Nesse capítulo é possível validar a eficiência do protótipo construído.

As conclusões e os trabalhos futuros são discutidos no capítulo 9

Enfim, o capítulo 10 lista as referências consultadas para a realização deste projeto.

2. DEFINIÇÕES IMPORTANTES

2.1 Considerações Iniciais

Neste capítulo serão apresentadas definições de termos importantes utilizados no texto deste projeto. Primeiramente, o conceito de *Web services* será elucidado. Em seguida, serão descritos os principais padrões de projetos e estruturas de dados utilizados na construção do protótipo. Por fim, o conceito de *pipeline* será apresentado.

2.2 *Web Services*

Segundo [1], *Web services* oferecem um *framework* extensível para comunicação aplicação-aplicação, construído a partir de protocolos *Web* existentes e baseado em padrões XML abertos e conhecidos.

Web services simplificam o processo de interação entre aplicações construídas em plataformas heterogêneas, definindo um mecanismo padrão para descrição, localização e comunicação com aplicações *Web.*

Geralmente, *Web services* são APIs *Web* que podem ser acessados pela Internet, e executados em sistemas remotos que hospedam os serviços. Por meio de *Web services*, uma aplicação expõe suas funcionalidades traduzidas em uma linguagem universal: o XML.

O *framework* de *Web services* pode ser dividido em três áreas: protocolos de comunicação, descrição de serviços e descoberta de serviços. Para cada uma das áreas são desenvolvidas especificações, as quais as mais utilizadas são:

- o protocolo SOAP (*Simple Object Access Protocol*) [2], provê um padrão para comunicação entre *Web services;*
- a linguagem WSDL (*Web Services Description Language*) [3], que provê uma descrição formal para *Web services,* em um padrão conhecido (XML)*;*
- o UDDI (*Universal Description, Discovery, and Integration*) [4], que provê um diretório para consulta à descrições de *Web services* disponíveis.

A Figura 1 ilustra a arquitetura de *Web services* construída com base nas especificações descritas acima.

Figura 1 - Arquitetura de *Web services* (adaptado de [5])

2.3 Padrão de Projetos DAO

O Padrão de Projetos DAO (*Data Access Object*) abstrai e encapsula o acesso às fontes de dados [6].

O modelo de negócio da aplicação utiliza uma interface única de acesso a dados. Diferentes implementações desta interface são usadas, de acordo com o tipo da fonte de dados. A implementação do DAO pode ser modificada sem alterações no restante do código, garantindo alto grau de desacoplamento entre a aplicação e a fonte de dados.

A Figura 2 mostra o diagrama de classes do padrão de projeto DAO. A classe *DataAccessObject* encapsula a lógica de acesso à fonte de dados, representada pela classe *DataSource*. Os objetos de negócio da aplicação (*BusinessObject)* usam os DAOs para obter *TransferObjects*¸ os quais representam as entidades de domínio.

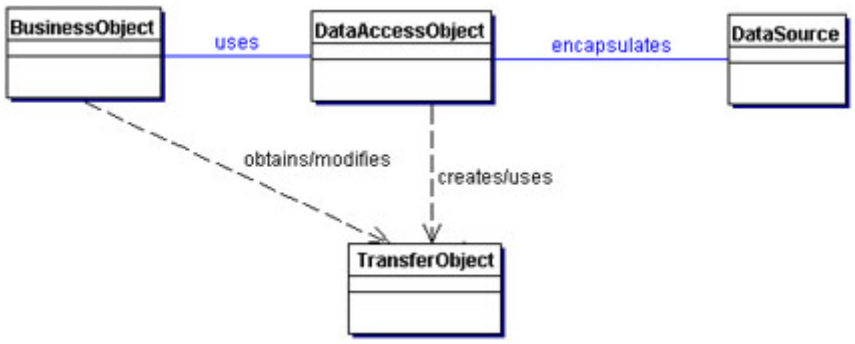

Figura 2 - Diagrama de classes do padrão DAO (fonte: [6])

2.4 Padrão de Projetos Model-View-Controller

Model-View-Controller (“modelo-visão-controle”) é um padrão de projetos arquitetural que organiza a aplicação em três módulos separados: um para o modelo da aplicação, incluindo a representação dos dados e a lógica de negócio. O segundo para a visualização, que provê a apresentação e a entrada de dados. E o terceiro para controlar requisições e o fluxo da aplicação [7].

Esse padrão provê inúmeros benefícios. Ele separa persistência, apresentação e controle, diminuindo a duplicação de código, centralizando o controle e fazendo a aplicação mais inteligível e fácil de ser modificada.

A Figura 3 mostra como acontece a comunicação entre os componentes no padrão MVC. O modelo, composto pelas entidades e lógica de negócio da aplicação, é modificado pelo controlador. Esse, por sua vez, é operado pelo usuário na camada de visualização. Como é possível observar, a camada de visualização não modifica dados diretamente no modelo.

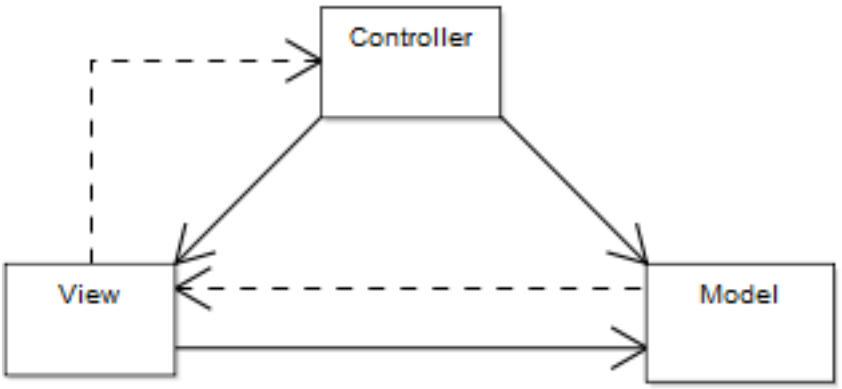

Figura 3 - Padrão Model-View-Controller (fonte: [7])

2.5 Padrão de Projetos *Pool de Threads*

Um *pool* de *threads* [8] é um padrão de projetos utilizado em sistemas que executam tarefas em paralelo. Esses sistemas geralmente disparam *threads* (ou processos) para cada tarefa a ser executada. Porém, caso o número de *threads* em execução simultânea se torne muito grande, pode haver uma queda no desempenho da aplicação. Isto se deve ao fato de que existe *overhead* de processamento para criação e destruição de *threads.*

Um *pool* de *threads* resolve este problema criando e disponibilizando apenas um número fixo e compatível de *threads* para execução das tarefas. Normalmente, o número de tarefas é muito maior que o de *threads.* Dessa forma, o *pool* organiza uma fila de tarefas para

execução de *threads.* Assim que uma *thread* do *pool* termina de executar uma tarefa, o *pool* utiliza essa *thread* para execução da próxima tarefa na fila. Esse processo se repete até que todas as tarefas na fila do *pool* tiverem sido executadas. Quando isso acontece, o *pool* destrói suas *threads* ou as coloca em estado *sleep.*

O número máximo de *threads* que o *pool* disponibiliza pode ser usado como um parâmetro para se otimizar o desempenho da aplicação. Com isso, evita-se saturar a memória do sistema, já que a utilização de memória virtual deixa o processo inaceitavelmente demorado por exigir escrita em disco. Assim, o *thread pool* é ajustado à configuração do *hardware* disponível.

2.6 Árvore (Estrutura de Dados)

Uma árvore é uma estrutura de dados definida por um grafo acíclico enraizado. Um nodo dessa árvore pode ter qualquer número de nodos filhos, e não há nenhum tipo de ordem entre esses [9].

Assim, as duas árvores mostradas na Figura 4 são iguais.

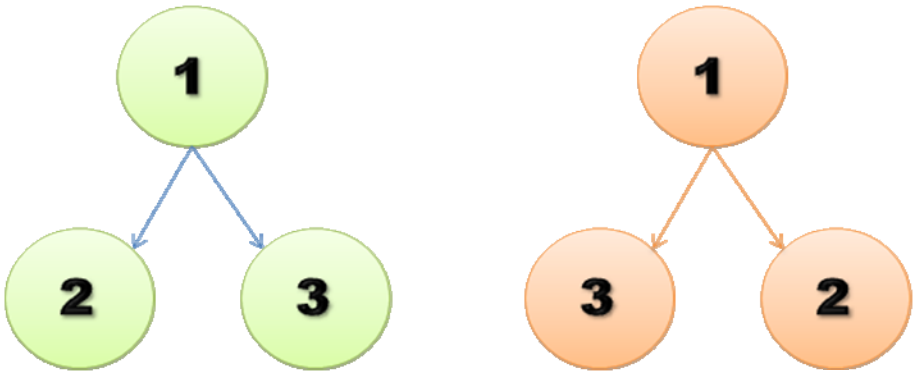

Figura 4 - Duas árvores recursivas iguais

A travessia em árvores pode ser feita recursivamente. Essas estruturas são muito úteis para a modelagem de estruturas hierárquicas, como árvores filogenéticas. Essa estrutura permite que processos em níveis elevados que utilizem etapas iniciais idênticas utilizem o processamento inicial já realizado, evitando desperdício de tempo computacional para a criação de *pipelines* que diferem apenas em passos finais.

2.7 Modelo de Arquitetura *Pipeline*

Pipeline é um modelo de arquitetura no qual ocorre a divisão de um processamento sequencial em etapas. A ideia fundamental desse modelo é que dados gerados pelo processamento de uma etapa sirvam como entrada para a próxima etapa, formando-se desse modo uma corrente de elementos de processamento.

O padrão de projetos arquitetural *Pipes and Filters* implementa *pipelines* em sistemas de *software*. Nesse padrão, os diferentes estágios de processamento são implementados por filtros. Os filtros são interconectados por canos (ou *pipes). Pipes* implementam o fluxo de dados entre passos adjacentes [10].

O pré-processamento efetuado em um pedido de compra exemplifica um *pipeline*, como pode ser visto na Figura 5. Três etapas de processamento são importantes para a "limpeza" do pacote de dados contendo o pedido. O pacote chega criptografado, por isso primeiramente passa por um filtro que remove a criptografia. No próximo passo, o pacote é autenticado, de forma a se verificar se o remetente é realmente quem diz ser. No terceiro e último passo, verifica-se por pacotes duplicados, uma vez que o usuário pode ter enviado dois pedidos iguais consecutivamente por engano. Na saída do *pipeline*, o pedido está pronto para ser processado.

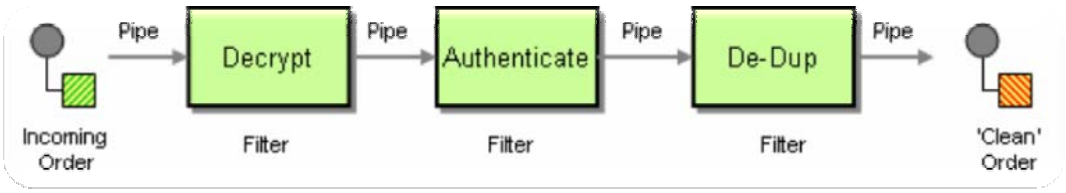

Figura 5 - *Pipeline* para pré-processamento de um pedido de compra (fonte: [10])

É importante ressaltar a diferença entre *pipelines* e *workflows*. Em *workflows*, geralmente podem ocorrer bifurcações na sequência de processamento. Por exemplo, em um determinado passo, o fluxo pode seguir por um caminho ou por outro. Dessa forma, deve-se tomar uma decisão em tempo de execução do *workflow*. Por outro lado, o fluxo de *pipelines*, como o próprio nome sugere, é linear. *Pipelines* funcionam como caixas fechadas, ou seja, depois que o mesmo foi configurado e iniciado, não é mais possível tomar decisões quanto à direção do fluxo de processamento. Apesar do caráter linear, é possível organizar *pipelines*

em árvores onde aqueles com primeiros passos repetidos sigam um mesmo caminho até divergirem. Essa é abordagem utilizada neste trabalho.

3. PIPELINES DE ANÁLISE GENÔMICA

3.1 Considerações Iniciais

O processo de análise do conteúdo genômico de uma espécie tem como objetivo a descoberta dos genes e suas funções. O processo inicia pelo sequenciamento de bibliotecas genômicas ou de expressão, o qual gera um grande volume de dados. O uso de ferramentas computacionais auxilia nas diferentes etapas de análise dos dados. *Pipelines* criam pontes de comunicação entre os diversos programas de forma a automatizar todo o processo.

Neste capítulo, o processo de análise genômica e os *pipelines* são explicados em maiores detalhes.

3.2 Processo de análise genômica

O processo de análise genômica é dividido em uma série de fases. A primeira etapa é a construção de bibliotecas, que podem ser genômicas ou de expressão (compostas por cDNA - DNA complementar ao RNA mensageiro). As bibliotecas constituem a fonte de onde se extrai o material genético a ser analisado.

Uma biblioteca genômica é composta por fragmentos de DNA da espécie estudada, e geralmente é portadora de moléculas representando todo o genoma do organismo. Por meio de ligação a vetores apropriados (plasmídeos, cosmídeos e outros) [11], esses fragmentos são inseridos em hospedeiros e clonados.

Bibliotecas de expressão também são construídas a partir da clonagem de hospedeiros contendo vetores. Porém, são compostas por clones de cDNA derivados de toda a população de RNAs mensageiros de uma célula ou tecido do organismo em estudo. Portanto, representam a expressão gênica da espécie em estudo. A escolha do tipo de biblioteca a sequenciar – genômica ou de expressão - depende dos objetivos do projeto. Inevitavelmente, genes raros apresentam dificuldade de amostragem em bibliotecas de expressão. Assim, de forma geral, quando o poder de sequenciamento é muito alto, opta-se por realizar o sequenciamento genômico, ou mesmo pela realização dos dois métodos. A utilização das duas estratégias pode gerar informações importantes relativas ao conteúdo genômico, expressão dos genes, processamento alternativo, regiões regulatórias, etc.

A segunda fase no processo de análise genômica é a determinação da sequência de nucleotídeos de cada um dos fragmentos constituintes das bibliotecas. O método mais

utilizado para esse fim é conhecido como Método de Sanger [12] automatizado. Esse método consiste na incorporação enzimática de nucleotídeos modificados a uma fita de DNA em formação, os quais impedem o crescimento da fita a partir do ponto em que são inseridos. Os nucleotídeos modificados possuem marcadores fluorescentes que permitem a identificação de suas bases através de cores. Através da leitura dos marcadores fluorescentes, é possível identificar a última base de cada fragmento truncado. Equipamentos conhecidos como **sequenciadores** realizam a análise do material sequenciado de forma automatizada, e são capazes de distinguir as cores e a posição de cada base na amostra. Os sequenciadores geram arquivos contendo dados brutos sobre as leituras de bases. Muitos autores se referem a esses arquivos pelo termo **cromatograma** (ou *chromat).* No entanto, a denominação correta seria fluorograma, uma vez que esses dados não contêm cores, e sim, medidas de fluorescência. Um cromatograma é, na verdade, um gráfico de análise fluorográfica da amostra codificado por cores, por meio do qual é possível visualizar a separação das bases da mistura. Apesar de formalmente incorreto, ao longo deste trabalho o termo cromatograma será utilizado para designar os arquivos gerados pelo sequenciador, devido à popularidade de seu uso. A Figura 6 ilustra um exemplo de cromatograma.

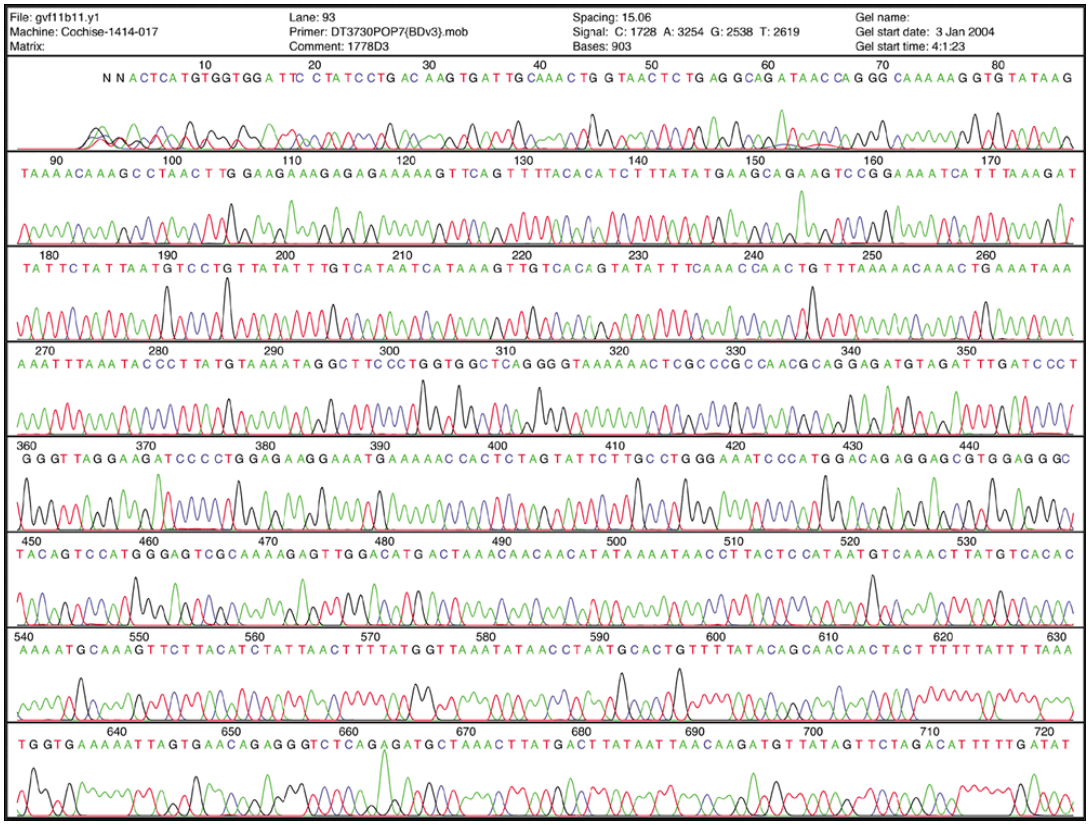

Figura 6 - Cromatograma gerado por sequenciador

Novos últimos anos, técnicas de sequenciamento mais eficientes que a de Sanger têm sido utilizadas com sucesso. Uma delas, conhecida como pirosequenciamento, é baseada na detecção de pirofosfatos liberados na incorporação de nucleotídeos durante a síntese de DNA. Atualmente, equipamentos que automatizam esse processo podem ler até 400 milhões de nucleotídeos em apenas 10 horas.

A partir desse ponto, torna-se essencial a utilização de sistemas computacionais. Os dados analógicos presentes nos cromatogramas são transformados em sequências de caracteres, que passam por uma série de estágios posteriores a fim de se descobrir os genes presentes e inferir outras informações biológicas.

Cada um desses estágios é executado por um programa computacional. O processo consiste em utilizar dados de saída de um programa como entrada para o seguinte [13]. Porém, executar cada um dos programas manualmente é extremamente trabalhoso, por isso, torna-se necessária a utilização de sistemas que automatizem todo o processo. Esses programas são denominados ***pipelines* de análise genômica**.

3.3 Etapas dos Pipelines de Análise Genômica

Pipelines de análise genômica são sistemas computacionais que executam sequencialmente uma série de programas de bioinformática, sendo que o resultado de um programa serve como entrada para o próximo programa na linha de execução. Geralmente, recebem como entrada um conjunto de cromatogramas gerados a partir de bibliotecas genômicas ou de expressão e retornam relatórios construídos a partir dos resultados dos programas.

Cada programa executa uma etapa do processamento de análise de sequências. A Figura 7 ilustra um *pipeline* típico de um projeto de sequenciamento genômico. As etapas de processamento são: nomeação das bases presentes nos cromatogramas (*base-calling*), limpeza das sequencias de baixa qualidade, agrupamento (*clustering*), montagem (*assembly*) dos fragmentos e anotação. As etapas são descritas com mais detalhes nas próximas seções, assim como os programas mais utilizados em cada uma delas.

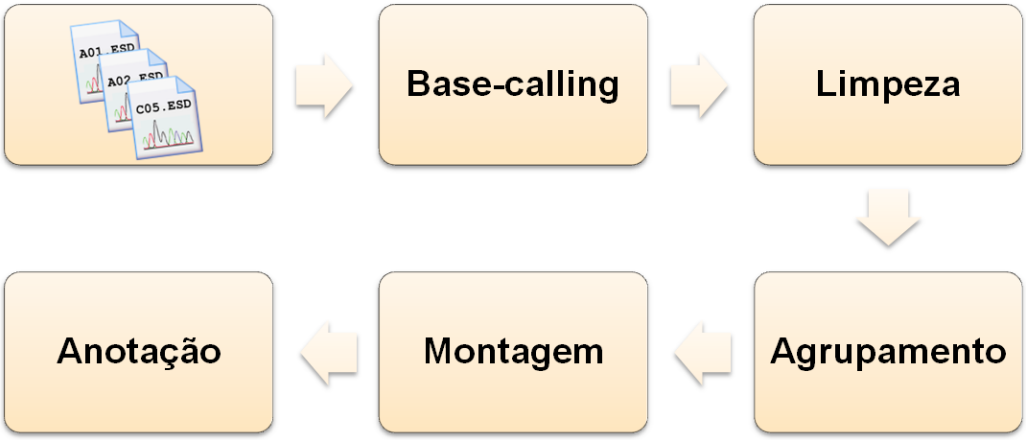

Figura 7 - Etapas comuns em um *pipeline* de análise de DNA ou cDNA

3.3.1 Nomeação de bases (Base-calling)

Na primeira etapa de um *pipeline* de análise genômica, os cromatogramas gerados pelo sequenciador passam por um processo denominado nomeação de bases ou *base-calling*. Nesse processo, é feito um cruzamento das informações de cores e acurácia dos picos nos gráficos do cromatograma, de modo a se determinar a natureza e a ordem das bases presentes na amostra, com certo grau de incerteza incorporado.

As bases identificadas são transcritas para arquivos de texto, normalmente em um formato conhecido como FASTA [14]. O cabeçalho desse arquivo contém o identificador da sequência. Na linha seguinte ao cabeçalho aparece a sequência de bases identificadas. Exemplos de sequências no formato FASTA podem ser vistas na Figura 8.

O principal programa para *base-calling* é *Phred* [15]. *Phred* tem sido usado extensivamente em laboratórios de sequenciamento em todo o mundo.

Juntamente com o arquivo FASTA contendo a sequência de bases, *Phred* gera um arquivo de qualidade, onde associa um valor de qualidade a cada base identificada. Esse valor está associado à probabilidade de erro na identificação de uma determinada base. O programa pode ser configurado para podar (*trimming* no inglês) o início e o final das sequências, os quais normalmente são locais onde o valor de qualidade fica abaixo dos níveis toleráveis.

```
>sequence1_name
ccccagctccggatgcaggcacaaacaacgtatggacaataggcactcttctaccaggtg
aatcaggaacaatagtcataaaagtgaaggtccctgaacagatagacctctcttttacag
aaacaggcagtgtaataggagaaggtgttgtaatggtcaccaaagacctatccacagagc
agaagccatattcgcttatgaatagcgttaccctatcctgtacagaactggctcctaaaa
gtgccttcgctaccacaaaagtgagtggcgtgcccggaacgagcctgggtattacagaaa
gcggcagtggcctctactcttctgacgagaacttggacttacagacaaagaacagaacta
tcatccttaagaagaccacgcatgcggaatacaaacccacttcttttaacttctc
>sequence2_name
tcaggtcctgcgtcacctataaaaaaaatcggtcttattgtgattgatgacagataatct
tcaactgtgcttataaatatagtattatcattcttgtttgcacttactgcgtcccagtaa
acaggtaacatccaccagtcatcgacaatac
>sequence3_name
ttacctaaaaaatctgaaactgttgtttgaaaaatcgaaattgggcgacttttgttcttt
ttaactttgagattttctgcgaaaagaaataggtatttgcttccacatgctgaagttact
tcataaccgtggtcgcatgttcttgtttttcgggttagaatgttctctatcggaattgtg
aaaaaatcaagacgagtagactgattaatagaaattatgcgaattttagtcgtttctttt
agtagttctaacctttttttcgaaaagtttttcatattgtttttgtccagatttacttttt
aaacttatagttattttacttacatttcctgaaatgtcgcttaattgagccaatatgtat
ttatatcctttgcacactatagtaatcatggattatgcccccattaaagttgtttatatc
attaaggccgattaaatactctaattggtataatcctacatatttctttttaaattgata
atttaaagcataaaaagcaaatttttagtggaatagtaactgtctatagaggtaattaaa
aagggagg
```

Figura 8 – Exemplo de três sequências no formato FASTA

3.3.2 Limpeza

O processo de *base-calling* gera milhares de sequências correspondentes aos fragmentos de DNA contidos nas placas extraídas das bibliotecas genômicas do projeto. O processo de limpeza é fundamental depois da identificação das bases, pois geralmente há sequências que apresentam DNA não pertencente à espécie estudada, regiões cujo sequenciamento é impreciso, e outras características não interessantes nas etapas subsequentes da análise.

A etapa de *base calling* gera um arquivo contendo valores de qualidade associados a cada base. A qualidade determina o grau de certeza de que uma base foi corretamente identificada. As bases que apresentam um valor de qualidade abaixo de um limite pré-estabelecido devem ser descartadas, uma vez que podem gerar imprecisões nas etapas subsequentes do *pipeline.*

As sequências geradas também podem conter contaminantes provenientes de fragmentos de DNA não pertencentes à espécie estudada. Esses fragmentos podem fazer parte de *primers*¸ vetores, adaptadores e/ou sequências de *Escherichia coli*.

Outras características possivelmente presentes nas sequências transcritas que podem dificultar o processamento das etapas subsequentes do *pipeline* são a presença de caudas *poli-A/poli-T* e repetições de elementos.

Na etapa de limpeza, os contaminantes, regiões de baixa qualidade, caudas *poli-A e poli-T* e repetições de elementos são removidos das sequências. Se depois desse processo o tamanho das sequências ficar abaixo de um limite pré-estabelecido, as mesmas são descartadas, não sendo utilizadas nas etapas subsequentes do *pipeline.*

Os parâmetros utilizados para limpeza podem variar bastante, de acordo com a espécie estudada ou com os objetivos dos pesquisadores. O limite mínimo para que uma base seja considerada de baixa qualidade e o tamanho mínimo para que uma sequência não seja descartada após a limpeza são exemplos de parâmetros configuráveis nessa etapa.

Três programas bastante utilizados nessa fase do *pipeline* são *RepeatMasker* [16], *cross_match* [17] e *SeqClean* [18]. *RepeatMasker* faz uma busca por elementos repetitivos em sequências de DNA. A saída do programa é uma descrição detalhada das repetições presentes nas sequências e uma versão modificada das mesmas na qual as repetições são mascaradas (bases substituídas por “Ns”).

SeqClean encontra e elimina contaminantes, regiões de baixa qualidade e caudas *poli-A/poli-T* de sequências. Para isso, efetua comparações em bases de vetores, adaptadores e outros contaminantes.

Cross_match utiliza o algoritmo de Smith-Waterman-Gotoh [19] para a realização de comparações em bases de dados de sequências, e é usado tanto na execução de *RepeatMasker* quanto de *SeqClean.* Alguns pesquisadores o utilizam em separado para mapear sequências de vetores, que são então excluídas.

3.3.3 Agrupamento e Montagem

Depois da execução da etapa de limpeza, têm-se então apenas sequências com um tamanho aceitável e com a qualidade desejada.

As sequências nesse momento são fragmentos de DNA ou cDNA obtidos aleatoriamente de uma biblioteca. Muitos deles são fragmentos de genes que foram quebrados no processo de clonagem ou regiões de sequenciamento parcial de moléculas maiores. Na etapa de montagem (ou *assembly)* do *pipeline*, os segmentos (contendo genes e/ou regiões intergênicas) são reconstruídos a partir dos fragmentos obtidos na etapa anterior.

A reconstrução de genes ou mesmo do genoma completo é possível, pois existem partes comuns entre diferentes fragmentos. Isso acontece, por exemplo, quando o final de um fragmento tem a mesma sequência de bases que o início de outro fragmento (sobreposição). Desse modo, o processo de reconstrução de genes e genomas é semelhante à montagem de um jogo de quebra-cabeça, todavia baseado em sobreposições parciais ao invés de encaixes. Quando existe sobreposição de bases entre dois ou mais fragmentos, esses formam um fragmento maior. O mapa das sobreposições é denominado ***contig.*** O **consenso** de um *contig* é formado pelas bases que mais parecem em cada posição (embora alguns se refiram ao consenso gerado também como *contig).*

Antes de iniciar a montagem de *contigs*, ou seja, extrair o consenso dos mapas de sobreposição, é necessário reunir em grupos (ou *clusters)* todos os fragmentos que possuem partes que se sobrepõem. Esse é um processo conhecido como **agrupamento** (também chamado pelo anglicismo **clusterização)**. O agrupamento normalmente é feito com o auxílio de programas de alinhamento múltiplo, como *MegaBLAST* [20]. Esse programa lê bases de dados de sequências e forma pares que possuem alto grau de identidade entre si, isto é, pares que possuem grande quantidade de bases nas mesmas posições. O grau de identidade mínimo utilizado normalmente é um valor acima de 96%. Isso torna muito pequena a chance de que o alinhamento entre as sequências de um par tenha ocorrido ao acaso. Portanto, dois fragmentos que formam um par têm uma alta probabilidade de compartilharem um segmento comum de um gene ou genoma, e por isso, são candidatos a formar um mesmo *contig*. Além da identidade entre os segmentos sobrepostos, utiliza-se uma análise de cobertura real/potencial dessa sobreposição. Como o alinhamento feito por *MegaBLAST* é local, exige-se geralmente nessa etapa que a região alinhada represente mais que 70% da sobreposição potencial das moléculas, o que é facilmente calculado sabendo-se o tamanho das regiões onde não houve alinhamento. Isso evita, por exemplo, que genes diferentes que compartilham um domínio idêntico sejam agrupados.

Para a formação de *clusters*, os pares de fragmentos formados pelo programa *MegaBLAST* e que passaram pelo filtro de cobertura descrito acima são utilizados em um algoritmo de agrupamento. Um algoritmo muito usado é *Single Linkage*, o qual coloca em um mesmo *cluster* todos os fragmentos que formam par com pelo menos um dos fragmentos já existentes no *cluster*. Por exemplo, se os fragmentos A e B formam um par, eles formam um *cluster*. Se um terceiro fragmento C forma par com o fragmento A, ele também faz parte desse *cluster*, não importando se C também forma um par com o fragmento B.

Dentro de um mesmo *cluster*, portanto, todo e qualquer fragmento tem sobreposição com pelo menos mais um fragmento do *cluster*. Assim, através da sobreposição de todos os membros de um *cluster* é possível montar um *contig*. A montagem do *contig* é representada pela geração do consenso dos fragmentos sobrepostos, isto é, no caso de mais de uma base ser encontrada em uma mesma posição, usa-se aquela que aparece mais vezes entre os fragmentos.

Quando um fragmento é o único membro de um *cluster*, esse dá origem a um ***singlet***, que nada mais é do que um *contig* composto por um único fragmento.

Para análises de genomas inteiros, o melhor caso seria a formação de apenas um *contig* contendo todo o DNA da espécie. Para o caso de análise de expressão gênica, é melhor encontrar *contigs* contendo o cDNA inteiro, ou pelo menos a região codificadora..

A Figura 9 ilustra o processo de montagem de *contigs*. As setas representam os fragmentos contidos em um *cluster*. O *contig* 1 é formado pela sobreposição dos fragmentos do mostrado à esquerda na figura. Na área circulada é possível notar que os fragmentos se alinham no ponto em que existe sobreposição. O mesmo acontece para o *contig* 2.

Três programas bastante utilizados para montagem são *CAP3* [21], *Phrap* [17] [22] e *TGICL* [23]. Esse último é efetua tanto agrupamento quanto montagem de fragmentos, e para isso, internamente faz uso de *MegaBLAST* e *CAP3*. Alguns projetos utilizam apenas *CAP3* para a formação de agrupamentos e montagem dos *contigs*. Embora *CAP3* não tenha sido desenvolvido para guiar a formação de *clusters*, ele acaba gerando *contigs* separados que equivalem a *contigs* de *clusters* diferentes.

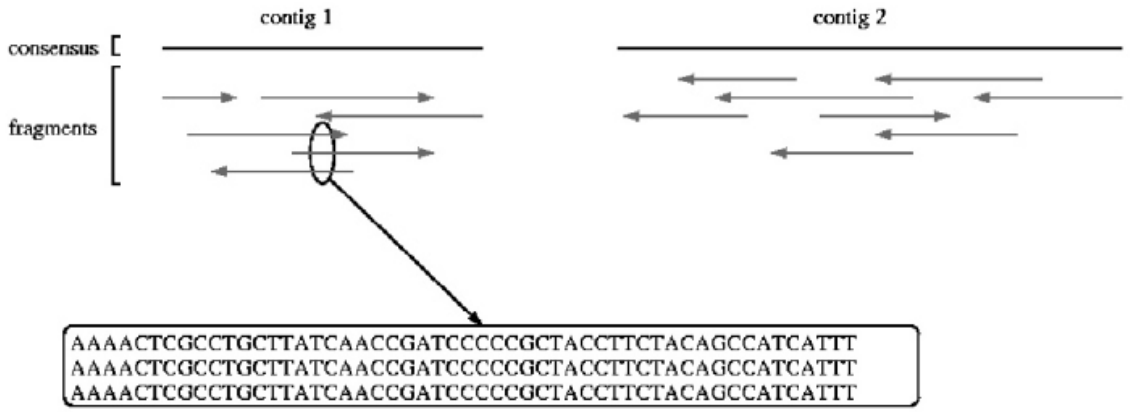

Figura 9 - Processo de montagem de *contigs*

3.3.4 Anotação

A etapa final de análise genômica consiste em um processo denominado **anotação**. Nessa etapa o pesquisador procura identificar os genes presentes nos *contigs* da etapa anterior, e também outras informações como as funções, participação em vias metabólicas e relações filogenéticas desses genes.

A anotação é, portanto, um processo de interpretação dos dados brutos gerados pelo sequenciamento, com o objetivo de acrescentar informações biológicas. Esse processo pode ser manual ou automatizado, sendo essa última forma realizada por sistemas de computação.

O processo de identificação de genes geralmente é feito através da comparação das sequências de cada *contig* ou *singlet* com genes já conhecidos, cujas sequências de nucleotídeos estão disponíveis em bancos de dados públicos. Nesses bancos, os genes podem estar representados na forma de suas sequências de nucleotídeos ou de proteínas. Bancos de dados como *PIR* [24], *Swiss-prot* [25] e *UniProt* [26] contém proteínas de vários organismos com anotação manual ou automática. Isso significa que informações biológicas provindas de estudos anteriores, como funções, vias metabólicas e relações filogenéticas, já estão associadas a cada uma das proteínas presentes nesses bancos.

Se há alto grau de alinhamento entre um *contig* e uma ou mais proteínas presentes nesses bancos, há grande probabilidade de que o *contig* contenha um gene **homólogo** (origem comum) aos encontrados no alinhamento. Desse modo, é provável que o gene codifique uma proteína com funções e características similares às daquelas encontradas no alinhamento. Para efetuar o alinhamento usa-se um programa de comparação de sequências de nucleotídeos com proteínas como *BLASTX* [27] . Esse programa faz primeiro a tradução, através do código genético, da sequência de nucleotídeos nas possíveis sequências de aminoácidos. O programa procura então alinhamentos das sequências protéicas obtidas em bases de dados de proteínas. O percentual mínimo de identidade entre as sequências para que o alinhamento seja considerado válido é configurável. É também possível configurar os valores mínimos para o *score* e máximo para o *e-value* em cada alinhamento válido. O *score* é calculado com base no número de lacunas e divergências entre bases em um alinhamento. Quanto maior o *score*¸ melhor é o alinhamento. O *e-value* representa o número de vezes que um alinhamento - ou algum melhor que ele - poderia ocorrer ao acaso em uma busca completa no banco de dados. Quanto menor o *e-value*¸ maior a chance da similaridade entre as sequências alinhadas serem explicadas por origem comum (homologia). A partir da configuração dos parâmetros, é

possível obter alinhamentos com probabilidade praticamente inexistente de terem ocorrido ao acaso (ou seja, sem relações de homologia).

3.4 Considerações Finais

Neste capítulo foi realizada uma revisão do conceito de *pipeline* de análise genômica e das etapas mais comuns de um *pipeline*, e dos principais programas utilizados nesses *pipelines*.

Sistemas computacionais têm sido de fundamental importância para o rápido crescimento do conhecimento relacionado aos genes e genomas de diversas espécies. O uso de *pipelines* auxilia e acelera o processo de análise genômica por meio da automatização das etapas de análise. A possibilidade de configuração de parâmetros em cada etapa permite uma maior adequação dos *pipelines* aos diferentes objetivos de projeto.

Para maiores informações sobre as etapas de *pipelines* de análise genômica, consultar [13].

4. TRABALHOS CORRELATOS

4.1 Considerações Iniciais

Na literatura é possível encontrar um grande número de trabalhos relacionados à *pipelines* para análise genômica. A maioria deles surge como fruto de projetos maiores, como grandes sequenciamentos genômicos, ou dentro de uma suíte mais ampla de ferramentas. Por isso, esses *pipelines* acabam sendo muito específicos e sem muitas alternativas de configuração. É o caso, por exemplo, de *ESTAP* [28], *GARSA* [29] e *ESTWeb* [30].

Algumas das ferramentas são complexos sistemas de *workflow*, como *BioPipe* [31] e *Pesasys* [32], nos quais é possível criar *pipelines* extremamente flexíveis. Por outro lado, exigem um conhecimento técnico muito maior por parte do usuário.

Nas próximas seções, serão dados detalhes de alguns dos principais *pipelines* disponíveis na literatura, suas características, vantagens e eventuais desvantagens. Pretende-se, dessa forma, justificar o sistema proposto neste projeto como alternativa relevante aos *pipelines* existentes.

4.2 Ensembl Pipeline

Ensembl Pipeline [33] é um *pipeline* construído na linguagem Perl com a proposta inicial de processamento dos dados do projeto *Ensembl* [34]. Mais tarde, foi disponibilizado à comunidade científica.

Esse *pipeline* adota um sistema de gerenciamento de *jobs (*tarefas) que são executadas dentro de um *compute farm: clusters* de computadores que processam um conjunto de *jobs* em paralelo.

No caso de *Ensembl Pipeline*, os *jobs* são configurados para executar *runnables*, que geralmente possuem *wrappers* contendo a lógica de um programa externo. Um gerenciador de regras (*RuleManager*) é utilizado para a verificação da ordem de precedência dos passos.

A entrada de dados de *Ensembl Pipeline* é um banco de dados relacional único, onde também são armazenados os parâmetros de cada programa.

Ensembl Pipeline constrói um *pipeline* eficiente que foi utilizado no sequenciamento de diferentes espécies como humano, rato, mosca e peixe-zebra. Porém, não é muito flexível. Novos módulos não podem ser inseridos com facilidade e não existem muitas opções de configuração.

Outro ponto negativo é que *Ensembl Pipeline* não apresenta uma interface gráfica para a entrada de dados.

4.3 Biopipe

Biopipe [31] é um redesenho da arquitetura de *Ensembl Pipeline*, mas em um sistema muito mais genérico. A intenção de *Biopipe* é funcionar como um *workflow* extremamente configurável.

Biopipe também é construído na linguagem *Perl* e utiliza um sistema de gerenciamento de tarefas derivado de *Ensemb Pipeline*. Também trabalha com uma *compute farm* para realizar o processamento paralelo de *jobs*. Outra semelhança é a presença do gerenciador de regras.

As tarefas em *Biopipe* são consideradas como análises individuais, sendo que cada uma possui um *status* de execução. O usuário que requisitou a análise pode efetuar o acompanhamento de seu andamento através desse *status.*

Biopipe tem muitos avanços em relação ao *pipeline* do qual derivou. Esse sistema introduz, por exemplo, o conceito de módulos reutilizáveis. Os pesquisadores podem construir *pipelines* personalizados em um documento XML no qual são configurados os módulos e os parâmetros a utilizar.

Também diferentemente de *Ensembl Pipeline*, *Biopipe* tem a vantagem de não estar restrito a apenas um sistema de armazenamento. O sistema permite configurar a entrada e a saída de dados através da implementação de classes de entrada e saída específicas.

Apesar de possibilitar grande flexibilidade, *Biopipe* adiciona bastante complexidade ao sistema. Determinadas configurações muitas vezes exigem conhecimentos avançados nas linguagens *Perl* e na biblioteca *Bioperl* [35].

4.4 Pegasys

Pegasys [32] é um sistema de *workflows* desenvolvido na linguagem Java.

Esse sistema permite a criação de *workflows* utilizando o conceito de DAGs (grafos acíclicos diretos). Esses são grafos que determinam a direção do processamento nas análises.

Através desse conceito, *Pegasys* consegue verificar a ordem de execução das análises, assim como a entrada e saída de cada uma delas. *Pegasys*, diferentemente de outros pipelines, permite que um passo do *pipeline* aceite mais de uma entrada. Também através do grafo, *Pegasys* infere que etapas podem ser executadas em paralelo.

O programa utiliza arquitetura cliente/servidor. No lado do cliente, *Pegasys* disponibiliza uma interface gráfica que permite que os usuários criem *workflows* com recursos intuitivos, tais como *drag and drop*. Os *workflows* são representados em documentos XML que são repassados ao servidor de *Pegasys*, onde são interpretados e executados.

No servidor, o sistema também utiliza o conceito de *cluster* de computadores para processamento paralelo de requisições. Também usa o modelo de *jobs* para execução de tarefas.

Pegasys disponibiliza ainda uma API (*Application Programming Interface)* para acesso ao banco de dados, que permite que os dados sejam acessados de forma programática, sem a necessidade do entendimento do modelo de dados do programa.

O uso de DAGs é eficiente no reaproveitamento da execução de programas. Os dados de entrada de diferentes requisições são reunidos e alimentam cada programa do *workflow*, que desse modo precisam ser executados uma única vez. Porém, não há reaproveitamento de resultados, já que as análises são totalmente independentes.

Pegasys de fato é uma ferramenta bastante flexível e permite a construção de complexos *workflows*. Porém, assim como em *Biopipe*, a complexidade faz com que o uso se torne difícil, e exija bons conhecimentos técnicos em programação por parte do usuário.

4.5 EGene

EGene [36] é um sistema desenvolvido em *Perl* que funciona de forma similar à *Pegasys*. Porém, apresenta uma interface mais simples e intuitiva.

Na interface do sistema, o usuário monta seu *pipeline* incluindo os diferentes programas e parâmetros a utilizar. O sistema então constrói um documento XML a partir desses dados e o envia ao servidor. O servidor interpreta o documento XML, executa o *pipeline* e retorna os resultados via email ao usuário. *EGene* suporta paralelismo quando a execução do *pipeline* se divide em diferentes subárvores.

Por causa da simplicidade, *EGene* é mais indicado para usuários que vão fazer análises pontuais. Em projetos com alto volume de dados, ou que dependam de submissões ao longo do tempo, a ferramenta não garante uma flexibilidade satisfatória.

EGene ainda permite a introdução de novos módulos pela implementação de uma interface na linguagem *Perl*.

4.6 ESTExplorer

ESTExplorer [37] é um *pipeline* construído na linguagem *Python* sobre a plataforma Zope [38] usando uma abordagem um pouco diferente dos demais. Esse *pipeline* utiliza processamento distribuído, onde as ferramentas da bioinformática são implementadas em processadores dedicados.

O sistema permite que o usuário escolha os programas a utilizar. O *pipeline* engloba um grande número de ferramentas incluindo programas para anotação funcional, ontologias gênicas, análise de padrões e mapeamento de vias metabólicas. Faz anotações funcionais tanto de nucleotídeos quanto de proteínas.

O processamento é dividido em três fases independentes podendo inclusive ser acoplado a outros *pipelines*.

A maior desvantagem de *ESTExplorer* é sua inflexibilidade. O sistema oferece poucas possibilidades de configuração dinâmica dos *pipelines*, não permite a inserção de novas bibliotecas de contaminantes, e a inclusão de novos módulos não é suportada.

O sistema ainda não possui suporte para controle de projetos.

4.7 EST2Uni

EST2Uni [39] é mais um sistema de análise genômica que possui anotação funcional e estrutural. É construído em *Perl* e PHP.

O sistema cria um *website* ligado ao banco de dados que permite consultas complexas e oferece ferramentas para mineração de dados, o que é o seu diferencial. Fornece também ferramentas de sistema de busca flexíveis.

A configuração do *pipeline* é feita através da edição de um arquivo texto. Esse é um fator que limita as possibilidades de configuração e impede que essas sejam feitas dinamicamente.

4.8 ChromaPipe

ChromaPipe [40] fornece uma abordagem mais interativa de análises genômicas. É voltado para análises individuais, de pequenas amostras. O usuário é capaz de interagir diretamente em cada passo do *pipeline*, modificando parâmetros e módulos a utilizar. Análises de programas como *CAP3*, *Phrap*, comparações do *BLAST* [27], *trimmings* e *vector maskings* podem ser feitos online.

Além disso, *ChromaPipe* permite que usuários individuais visualizem seus históricos de análise, acessem os dados submetidos, e apliquem algumas análises pós-sequenciamento básicas. *ChromaPipe* também apresenta interfaces para consultas de resultados.

A maior limitação do uso de *ChromaPipe* é o processamento do grandes volumes de dados. Nesses casos, o programa não oferece muitas opções de configuração do *pipeline*.

4.9 Magic-SPP

Magic-SPP [41] é um sistema de análise genômica construído em *Perl*. As interfaces de interação com o usuário são construídas em JSP e Java.

Magic-SPP possui um grande diferencial, pois inclui uma interface para administração dos dados de laboratórios (LIMS). Dessa forma, o sistema se beneficia com funcionalidades como o controle de projetos, usuários, amostras, instrumentos, padrões, e outros.

O sistema também inclui interfaces para controle de qualidade e resolução de problemas.

A submissão de cromatogramas fica interligada ao sistema de controle do laboratório, de forma que se torna mais fácil rastrear a origem e outras informações importantes a respeito dos cromatogramas submetidos.

Em *Magic-SPP*, o banco de dados não serve apenas como um repositório de dados, mas também controla o processamento de cromatogramas.

Apesar dessas inovações, os *pipelines* são pouco configuráveis, e o sistema não dá suporte à inclusão de novos módulos.

4.10 ESTIMA

ESTIMA [42] é um sistema de análise genômica com o diferencial de controlar múltiplos projetos e suportar múltiplas bibliotecas em cada projeto.

O usuário também pode, de modo limitado, personalizar suas análises. É possível, por exemplo, fazer buscas e comparações em genomas de referência selecionáveis, e escolher entre diferentes opções de algoritmos de agrupamento e montagem.

O esquema de banco de dados de ESTIMA é flexível e aceita como entrada inclusive os resultados de outros *pipelines*.

No pacote de ESTIMA também estão incluídas uma série de ferramentas *Web* que adicionam um conjunto de novas funcionalidades ao sistema.

ESTIMA possui uma interface *Web* dividida em diferentes aplicações, que permite diversas interações com o banco de dados.

ESTIMA utiliza arquivos de configuração na linguagem XML para guardar informações específicas dos projetos que controla.

4.11 Outros Pipelines

ESTWeb [30] é um sistema simples que se propõe a efetuar as etapas básicas de um *pipeline* de análise genômica: recepção de cromatogramas, etapas de *base-calling*, *screening* e comparação com bancos de dados públicos, armazenamento de dados, análises em bancos de dados relacionais e exibição de relatórios com estatísticas de produtividade e redundância.

ESTWeb implementa essa proposta fornecendo uma interface *Web* simples para envio de cromatogramas e exibição de relatórios. O sistema foi usado com sucesso no projeto de sequenciamento de *Schistosoma mansoni.*

Em contrapartida à simplicidade de uso, o sistema não é escalável, e o *pipeline* não é configurável, isto é, os parâmetros pré-definidos em cada programa não podem ser modificados.

Um sistema com uma proposta muito parecida a *ESTWeb* é *ESTAP* [28]. *ESTAP* também apresenta interface *Web* para submissão de cromatogramas, efetua um *pipeline* não-configurável e exibe relatórios de execução.

PartiGene [43] é um *pipeline* simples, escrito em *Perl*, que permite atualizações incrementais de bancos de dados genômicos. A entrada de dados acontece por linha de comando. O sistema não apresenta interface gráfica. Esse sistema gera um documento HTML contendo os resultados das análises.

PipeOnline 2.0 [44] é mais um *pipeline* simplificado, sem muitas opções de configuração de parâmetros e inclusão de novos módulos. Permite customização para projetos individuais. Uma das vantagens sobre os demais pipelines é que *PipeOnline* inclui análise funcional das sequências.

4.12 Discussão

Os sistemas de análise genômica estudados têm mostrado eficiência na análise de genomas e ESTs. Muitos deles foram e continuam sendo usados com sucesso em grandes projetos de sequenciamento.

Como pôde ser observado no estudo, a maioria desses sistemas surgiu em projetos de sequenciamento com a finalidade de atender as demandas individuais de cada projeto. Por isso, não houve uma preocupação com a extensibilidade e com a possibilidade de configurações dinâmicas de *pipelines*.

A falta de pontos de extensão é uma deficiência em sistemas desta natureza. Programas incluindo análises novas e mais eficientes têm surgido constantemente. Sistemas de análise genômica que não permitem a inclusão e configuração de novos módulos não podem se beneficiar com a inclusão desses novos programas, e por isso suas análises tendem a se tornarem obsoletas. É o caso de sistemas como *ESTAP*, *PartiGene* e *ESTWeb*, que deixam de efetuar etapas importantes da análise e não podem ser estendidos de modo a receber novas funcionalidades.

Por outro lado, pode-se notar que o excesso de pontos de configuração e extensão aumenta a complexidade tanto do código quanto da usabilidade dos programas. Em sistemas como *BioPipe* e *Pegasys*, por exemplo, é possível construir *workflows* flexíveis. Entretanto, é necessário um bom conhecimento técnico de programação, tornando o uso inviável para boa parte dos pesquisadores.

O estudo também revelou que a maioria dos sistemas tem maior enfoque em análises pontuais do que no acompanhamento de projetos. Em grandes projetos de sequenciamento, é de extrema importância que os dados submetidos possam gerar resultados parciais para que o andamento dos projetos seja acompanhado. Informações sobre os laboratórios e pesquisadores que submeteram os dados podem ser de grande importância em projetos maiores. O único sistema que demonstrou tal funcionalidade foi *Magic-SPP*.

4.13 Considerações Finais

Apesar da existência de diversas ferramentas para a análise genômica, o estudo realizado demonstrou que muitos pontos podem ser explorados e estendidos.

O fator que ficou mais evidente é a falta de ferramentas para acompanhamento de projetos e colaboração entre os pesquisadores. Nos últimos anos, temos visto um avanço muito grande de tecnologias da denominada *Web 2.0*. Essas tecnologias colocam em foco a colaboração, isto é, a contribuição de cada usuário individual dentro da construção de redes de informação [45].

A possibilidade de configuração de *pipelines* é outro ponto pouco explorado na maioria dos sistemas estudados. Os objetivos do projeto podem requerer que diferentes

parâmetros e programas sejam utilizados nas análises. Tal fato gera a necessidade de *pipelines* configuráveis.

O uso de múltiplos *pipelines* em um projeto também é desejável, uma vez que diferentes grupos de pesquisa ou pesquisadores podem divergir quanto aos parâmetros e programas a utilizar. A comparação entre os resultados de múltiplos *pipelines*, cada qual com uma configuração diferente, pode gerar uma discussão sadia quanto aos melhores parâmetros e programas a utilizar no sentido de se atingir os objetivos do projeto.

Neste trabalho, um protótipo de *pipeline* foi desenvolvido com o intuito de suplantar as deficiências encontradas nos sistemas existentes. O protótipo, descrito nos capítulos 5 e 6 inclui suporte a acompanhamento gradual de projetos, múltiplos *pipelines* e configuração de programas e parâmetros nos *pipelines*, entre outras funcionalidades. O protótipo também foi construído com base em *Web services*, no sentido de desacoplar a dependência entre interfaces gráficas específicas e o processamento da análise, além de permitir que outros sistemas — possivelmente construídos em plataformas diferentes — possam se comunicar e efetuar análises.

5. PROTÓTIPO

5.1 Considerações Iniciais

Como descrito na introdução deste documento, a proposta deste trabalho é a apresentação de um protótipo de *pipeline* de análise genômica. Além das funções de análise das sequências brutas geradas pelo sequenciador, o protótipo também deve implementar os seguintes requisitos: análise progressiva de dados submetidos, suporte a múltiplos projetos, execução paralela de *pipelines*, reaproveitamento de passos e extensibilidade do código.

Neste capítulo é descrita a metodologia utilizada no protótipo construído neste trabalho. Nas sessões iniciais são apresentadas uma visão geral do sistema, a arquitetura utilizada e o fluxo de funcionamento da aplicação. Nas sessões seguintes, a lógica utilizada para a realização das diferentes funcionalidades é descrita com maiores detalhes. Este capítulo não apresenta detalhes técnicos da implementação do protótipo. Esses detalhes são fornecidos no capítulo 7.

5.2 Visão Geral

O protótipo construído neste trabalho é um sistema para análise de cromatogramas gerados em projetos de sequenciamento genômico. O sistema aceita como entrada arquivos compactados contendo conjuntos de cromatogramas gerados em sequenciadores. Os cromatogramas recebidos são analisados em *pipelines* constituídos de uma série de etapas de processamento.

Cada etapa de um *pipeline* é executada por um módulo que tem uma função específica no processo de análise genômica. O primeiro módulo converte cromatogramas em sequências de bases enquanto o último busca informações biológicas das sequências encontradas. Ao final do processamento, relatórios são gerados contendo estatísticas e resultados das análises.

Os *pipelines* são executados periodicamente ao longo do desenvolvimento do projeto genômico. A cada execução, os novos cromatogramas recebidos são processados e resultados parciais são gerados. Desse modo, o pesquisador fica constantemente informado sobre o estado atual do projeto.

O sistema é descrito neste capítulo através de seus casos de uso. Cada um deles representa uma etapa do fluxo de funcionamento da aplicação. Os casos de uso são listados a seguir.

- Gerenciamento de Dados de Projeto
- Criação e Configuração de *Pipelines*
- Submissão de cromatogramas
- Execução de *pipelines*
- Visualização de resultados

Nas próximas seções deste capítulo, os casos de uso explicados com maiores detalhes. A próxima seção apresenta o fluxo de funcionamento do sistema.

5.3 Fluxo de Funcionamento

O esquema apresentado na Figura 10 mostra o fluxo de funcionamento do sistema.

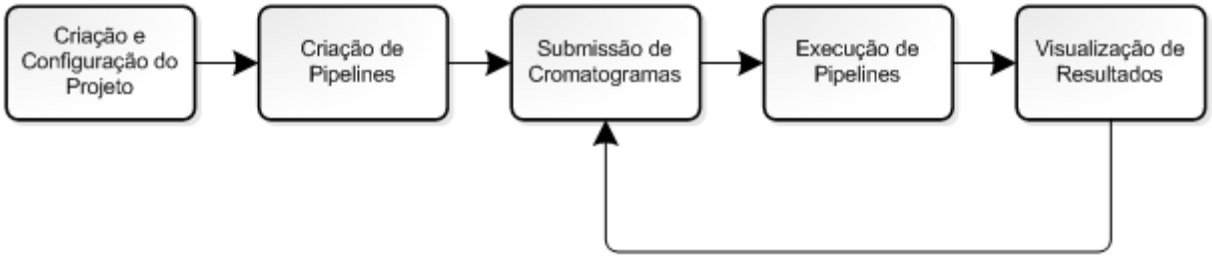

Figura 10 - Fluxo de funcionamento do sistema

Para a execução de *pipelines* é primeiramente necessário configurar dados do projeto. É preciso, por exemplo, definir os laboratórios participantes e nomear as bibliotecas genômicas utilizadas. Outro passo importante é informar os possíveis contaminantes presentes nas amostras.

A segunda etapa é a criação e configuração de *pipelines*. Nesta etapa, o pesquisador deve definir os parâmetros a se utilizar em cada programa que compõe o *pipeline*. É possível também que, para um mesmo projeto, o pesquisador crie diversos *pipelines* com diferentes configurações, com o objetivo de posteriormente comparar os resultados obtidos.

Depois de configurar os dados do projeto e os *pipelines*, o pesquisador pode iniciar a submissão de cromatogramas. Cada submissão deve conter todos os cromatogramas obtidos a partir de uma placa sequenciada. O sistema então executa os *pipelines* sobre os novos

cromatogramas recebidos. A execução de *pipelines* é periódica, sendo que o intervalo entre as execuções é configurável.

A qualquer momento, o pesquisador pode visualizar relatórios gerados a partir dos resultados parciais dos *pipelines* executados sobre as submissões efetuadas. Novas submissões também podem ocorrer a qualquer momento, reiniciando o ciclo de execução de visualização de resultados.

As lógicas utilizadas para a realização destes casos de uso são descritas com maior profundidade nas próximas seções deste capítulo.

5.4 Arquitetura

O protótipo é dividido em três componentes: *Núcleo, Web Service* e *Web*. No componente Núcleo são realizadas todas as operações de lógica e as transações com o banco de dados. As lógicas de criação de projetos, e de execução de pipelines, por exemplo, estão presentes no Núcleo.

Web Service serve como um *front end* para as operações de lógica do sistema. O componente oferece uma interface para comunicação entre aplicações remotas e o Núcleo, por meio de um serviço *Web (*em inglês, *Web service)* com operações que permitem o envio de dados e consultas. A submissão de cromatogramas, por exemplo, é realizada através do *Web service*.

O uso de *Web services* oferece duas vantagens importantes:

- Permite a comunicação, por meio de uma linguagem universal, entre sistemas construídos em plataformas diferentes. Desse modo, o servidor de processamento não fica acoplado a implementações específicas do cliente, isto é, clientes construídos em plataformas heterogêneas podem acessá-lo.
- Faz com que a lógica e os dados do sistema sejam independentes da interface visual, permitindo que os dados e o processamento sejam executados em servidores seguros e dedicados.

O componente *Web* é uma aplicação *Web* composta por formulários e interfaces gráficas. Esta aplicação cria um ambiente "amigável" e auto-explicativo onde o pesquisador dispõe de funcionalidades para operar o *pipeline*. A interação entre o componente *Web* e o Núcleo do sistema é possível através de chamadas às operações presentes no *Web Service*.

A Figura 11 mostra a arquitetura do sistema, onde é possível visualizar a relação entre os casos de uso e os componentes do sistema.

Os componentes Núcleo e *Web Service* se localizam no servidor de processamento enquanto o componente *Web* fica no servidor *Web*.

Percebe-se nessa figura que as funcionalidades que exigem interação do usuário - criação de projetos e *pipelines*, submissão de cromatogramas, e visualização de relatórios - são disparadas pelo usuário no componente Web e transmitidas ao Núcleo por meio de acesso do *Web Service*. A execução de *pipelines* é realizada periodicamente no Núcleo. O Núcleo é o único dos componentes com acesso ao banco de dados do sistema.

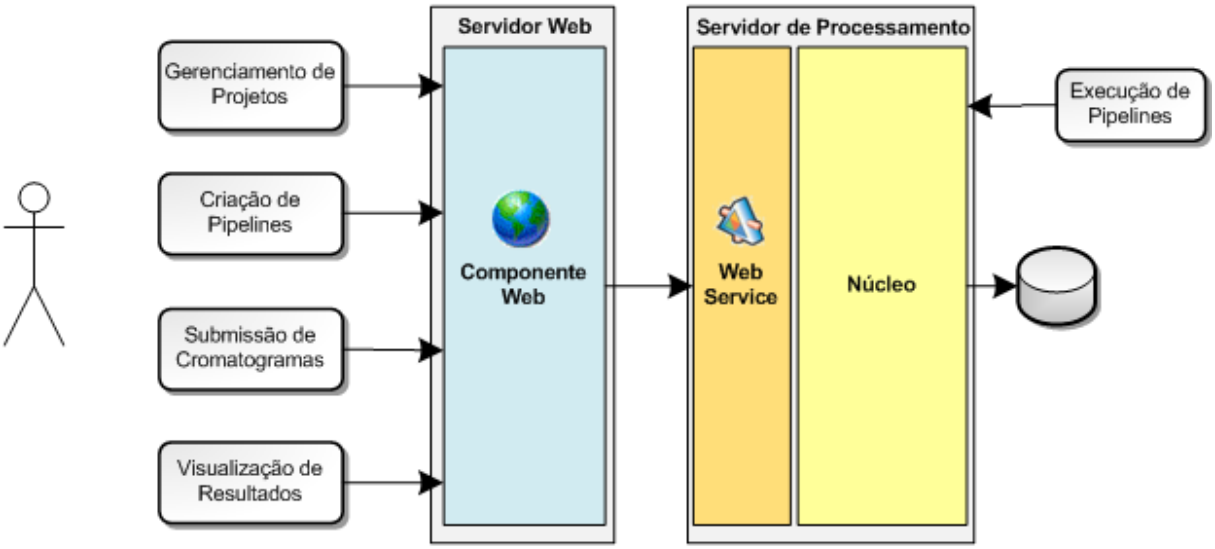

Figura 11 - Arquitetura do Sistema

5.5 Gerenciamento de Dados de Projeto

O sistema construído neste projeto armazena dados sobre os projetos em andamento. Isso é necessário, pois os resultados parciais das análises são armazenados. Portanto, a configuração de um projeto genômico no sistema é o primeiro passo para a sua utilização. Todos os demais passos dependem dos dados informados nesse primeiro.

Na configuração inicial de um projeto no sistema, deve-se informar: dados técnicos sobre o projeto (nome, *website*, nome do líder, email do líder, etc.); os laboratórios participantes; as bibliotecas genômicas utilizadas; possíveis contaminantes (vetores, *primers*, etc.) e a chave de acesso ao projeto.

Os laboratórios e as bibliotecas são identificados por códigos únicos de três letras. Estes códigos devem ser informados nas submissões de cromatogramas, como é detalhado na seção 5.7.

A chave de acesso é um identificador para o projeto. Como o sistema tem suporte a múltiplos projetos, a chave de acesso deve ser única entre eles. A chave é usada em todas as chamadas feitas ao *Web service*, servindo como um meio de identificação do projeto.

Os três componentes do sistema participam no gerenciamento de dados do projeto. No componente Web, o sistema disponibiliza formulários para que o pesquisador possa entrar com dados do projeto, laboratórios, bibliotecas e contaminantes.

A Figura 12 mostra o formulário usado para criação de projetos. A Figura 13 exibe o formulário de inserção de novos laboratórios. O formulário para inserção de bibliotecas genômicas é apresentado na Figura 14.

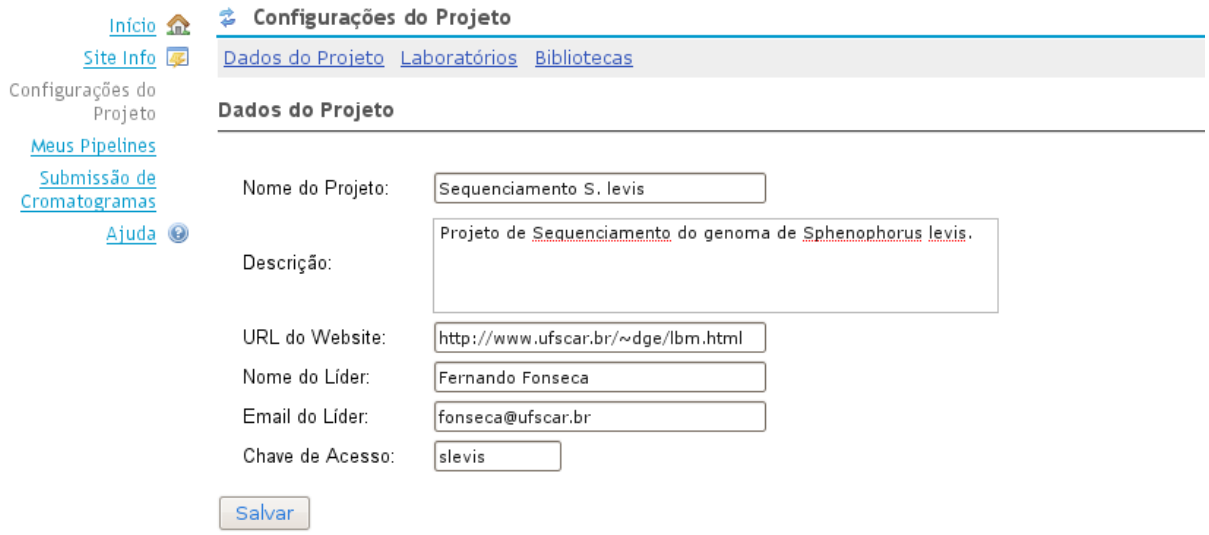

Figura 12 - Formulário para criação e edição de projetos

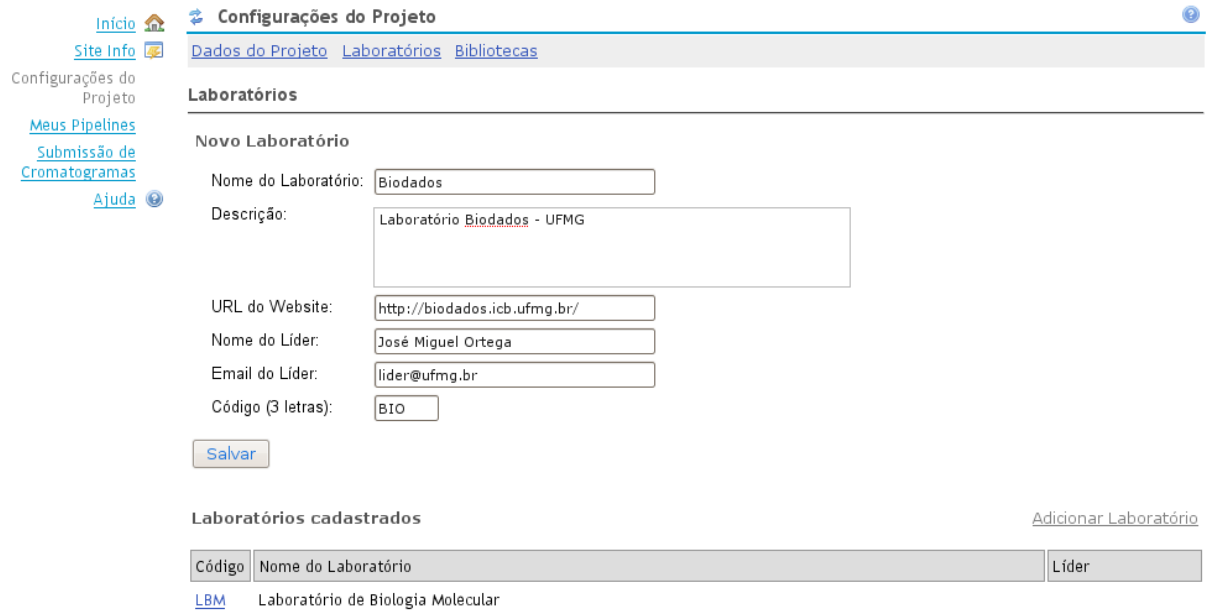

Figura 13 - Formulário para inserir novo laboratório

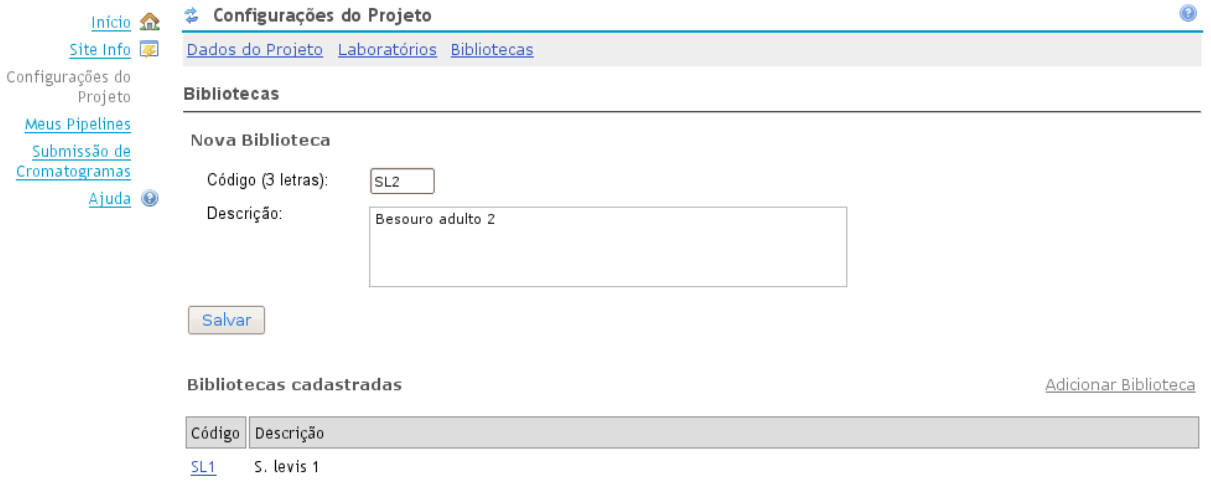

Figura 14 - Formulário para inserir novas bibliotecas

Quando os formulários são submetidos, o componente *Web* chama operações específicas para gerenciamento de projetos no componente *Web Service*. Estas são as operações:

- Criar/alterar projeto: operação cujo objetivo é criar um novo projeto no sistema ou alterar um projeto existente. Deve-se informar os principais dados do projeto, como nome, descrição, *website* e líder. Para a criação de um novo projeto, é necessário se informar uma chave de acesso única (não utilizada em outros projetos).

- Adicionar laboratório a projeto: deve-se informar nome, descrição e outros dados do laboratório e também a chave de acesso do projeto no qual o laboratório participa. O laboratório é identificado por um código de três letras.

- Adicionar biblioteca genômica a projeto: deve-se informar o código e a descrição da biblioteca (por exemplo, "BA1 - Besouro Adulto 1"). Também é necessário informar a chave de acesso do projeto.

- Adicionar sequências contaminantes: pode-se informar possíveis contaminantes das bibliotecas genômicas do projeto, como vetores e *primers*, os quais devem ser incluídos em formato FASTA. Esses contaminantes são utilizados pelo *pipeline* na etapa de limpeza, quando são removidos das sequencias encontradas (ver seção 6.3). Outra possibilidade é determinar que *UniVec* [46] seja utilizado como base de contaminantes.

O *Web Service* tem acesso direto à interface de lógica (API) do componente Núcleo. Para cada uma das operações listadas acima, existe uma operação correspondente na API de lógica do núcleo. Nesse componente existem métodos para armazenamento do projeto no banco de dados, juntamente com os laboratórios, bibliotecas e contaminantes.

A Figura 15 mostra o modelo lógico de dados utilizado para o gerenciamento de dados de projeto. A tabela *GENOME_PROJECT* armazena os dados do projeto e pode estar associada a zero ou mais registros das tabelas *LABORATORY* e *LIBRARY*, que armazenam informações sobre laboratórios e bibliotecas genômicas, respectivamente. A tabela *CONTAMINANT_DB* armazena informações sobre os contaminantes de cada biblioteca. *PROJECT_PROPERTY* guarda propriedades específicas de cada projeto.

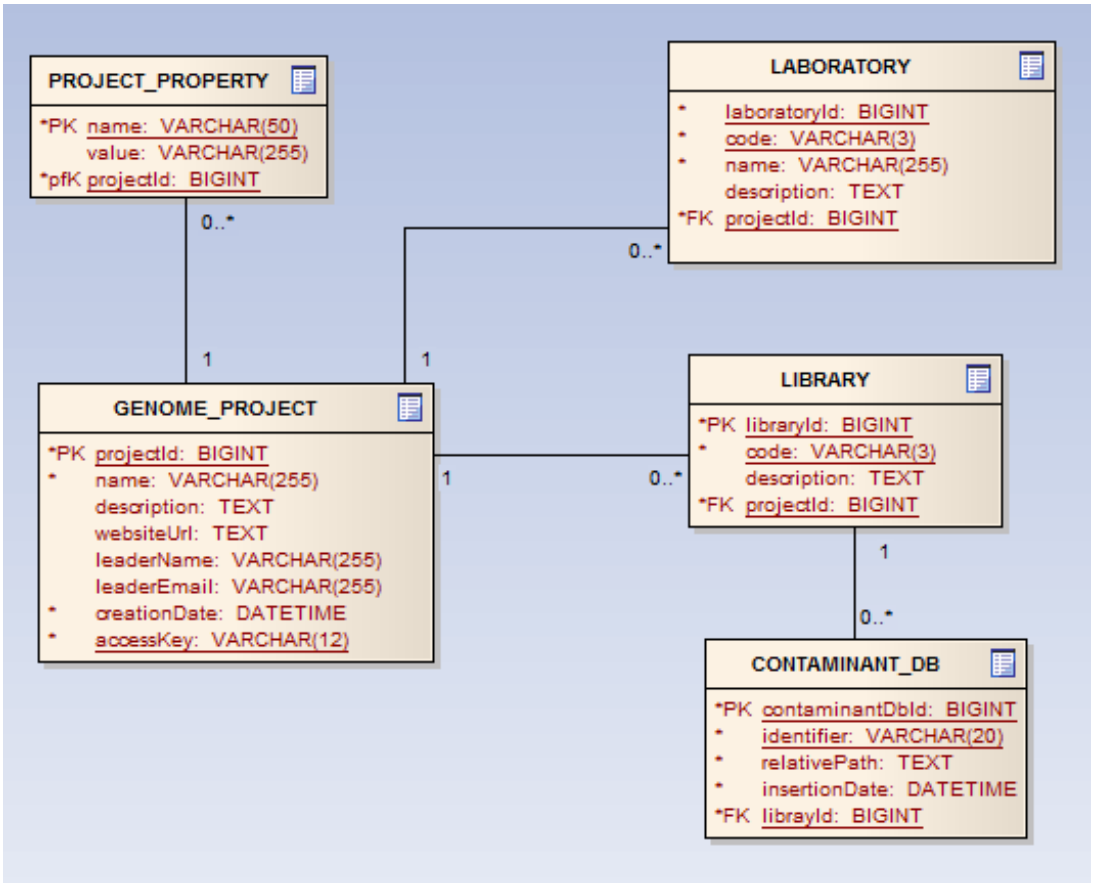

Figura 15 - Modelo lógico de dados para gerenciamento de projetos

5.6 Criação e Configuração de *Pipelines*

No contexto deste trabalho, um *pipeline* é definido como uma sequência de passos a serem executados sobre os dados presentes no banco. Um passo está sempre associado a um **módulo**. A execução de um passo é definida como a execução do módulo ao qual está associado, a partir de um conjunto de parâmetros.

Neste protótipo, cinco módulos estão definidos, sendo um para cada etapa do processo de análise de cromatogramas: módulo *Phred*, responsável pela etapa de *base calling* (descobrimento da ordem e natureza das bases); módulo *SeqClean*, responsável pela limpeza das sequências de baixa qualidade; módulo *Clustering*, responsável pelo agrupamento em *clusters* de sequências com alto grau de identidade entre si; módulo *CAP3*, responsável pela montagem de *contigs* a partir dos *clusters*; módulo *Pathways*, responsável por descobrir funções e vias metabólicas dos genes presentes nos *clusters* encontrados. É possível observar que as lógicas dos módulos geralmente dependem da execução de programas externos. Os módulos são explicados com maior profundidade no capítulo 6.

A comunicação entre dois passos consecutivos é feita via banco de dados. Desse modo, um módulo lê seus dados de entrada do banco de dados, e após sua execução os resultados gerados são novamente armazenados no banco, para que sirvam de entrada ao próximo módulo na sequência do *pipeline*. A Figura 16 mostra a comunicação entre módulos.

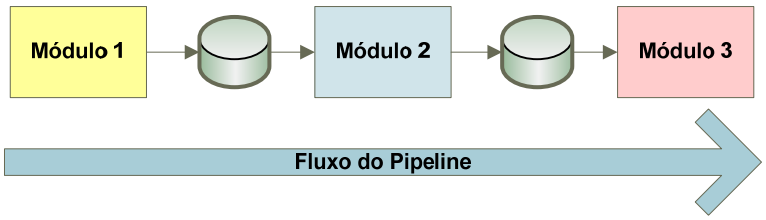

Figura 16 - Comunicação entre módulos

Uma das características do protótipo construído neste trabalho é que os *pipelines* são configuráveis. O pesquisador deve, antes de iniciar a submissão de cromatogramas, criar e configurar os *pipelines* que serão executados. É importante notar que em um mesmo projeto podem ser criados inúmeros *pipelines*, cada um com uma configuração diferente. Este é um recurso importante, uma vez que a variação de parâmetros de análise gera resultados finais diferentes. Cabe a cada grupo de pesquisa escolher parâmetros adequados aos seus objetivos.

A configuração de um *pipeline* define os parâmetros a serem utilizados nos módulos associados a cada um de seus passos. O sistema provê um formulário presente no componente *Web* através do qual o pesquisador pode criar e configurar *pipelines*. A Figura 17 mostra o formulário utilizado para criação de *pipelines*. É possível observar na figura que para cada etapa do *pipeline* a ser criado existe um conjunto de parâmetros a se definir.

Quando ocorre a submissão do formulário de criação de *pipelines*, o componente Web faz uma chamada a uma operação no *Web Service* denominada *createPipeline*. Esta operação recebe como parâmetro, além da chave de acesso do projeto, um documento XML contendo a configuração do *pipeline* a ser criado. Os documentos XML de configuração de pipelines devem estar em um formato reconhecido e validado pelo sistema. A Figura 18 ilustra um exemplo de documento XML de configuração de *pipeline*.

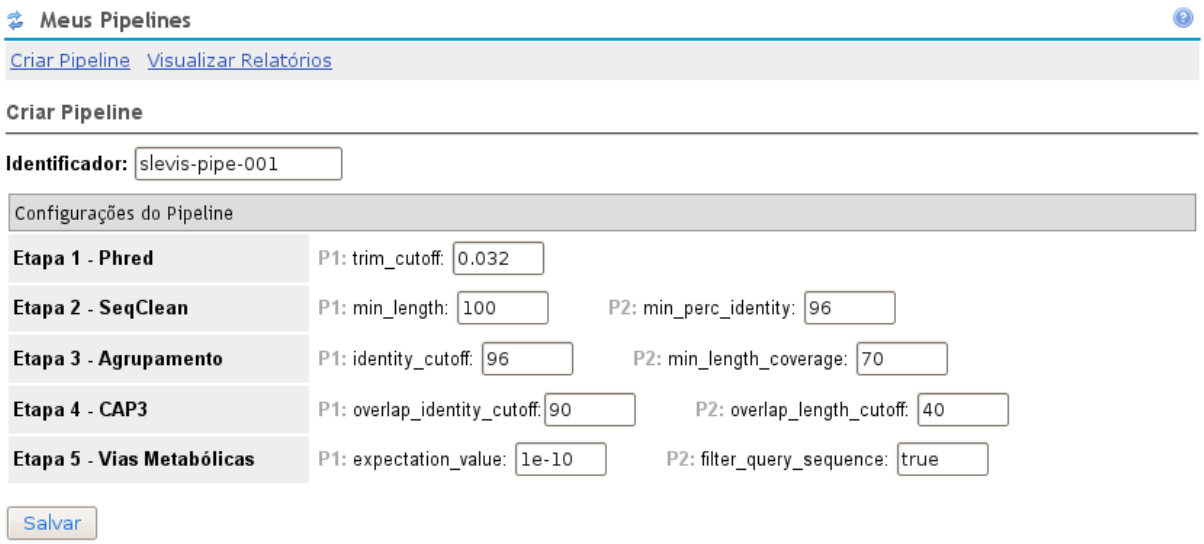

Figura 17 - Formulário para criação de *pipelines*

No exemplo da Figura 18, o elemento raiz é <*pipe_config*>. Este elemento denomina a configuração a ser usada no novo *pipeline* através de um identificador único (atributo *id*). Esse identificador é de fundamental importância, pois é responsável por identificar os resultados do *pipeline* após o processamento. O elemento <*pipe_config*> é composto por uma série de elementos <*module*>. Cada elemento <*module*> representa um módulo a ser executado pelo *pipeline*, na ordem de passos apresentada no arquivo de configuração. Os módulos são identificados pelo atributo *name*. Cada elemento <*module*> possui zero ou mais elementos <*param*>. Esses elementos representam os parâmetros de cada módulo, sendo o atributo *name* o nome do parâmetro. O valor do parâmetro é lido do texto entre as *tags* <*param*> e </*param*>. Na Figura 18, por exemplo, o módulo *phred* receberá o parâmetro *trim_cutoff* com valor 0.16. O módulo *clustering* recebe o valor 70 para o parâmetro *min_length_coverage.* O mesmo vale para os outros módulos e parâmetros descritos no arquivo de configuração.

```
<pipe_config id="pipeconfig_001">
    <module name="phred">
        <param name="trim_cutoff">0.16</param>
    </module>
    <module name="seqclean">
        <param name="min_length">80</param>
        <param name="min_perc_identity">45</param>
    </module>
    <module name="clustering">
        <param name="identity_cutoff">95</param>
        <param name="min_length_coverage">70</param>
    </module>
    <module name="cap3">
        <param name="overlap_length_cutoff">30</param>
        <param name="overlap_identity_cutoff">92</param>
    </module>
    <module name="pathways">
        <param name="expectation_value">1e-10</param>
    </module>
</pipe_config>
```

Figura 18 - Documento XML de configuração de *pipeline*

5.6.1 Montagem da árvore de pipelines

Assim que recebe uma configuração válida, o componente *Web Service* chama a API lógica do Núcleo para o registro do novo *pipeline* no sistema.

No componente Núcleo, os *pipelines* poderiam ser simplesmente armazenados individualmente no banco de dados, sem nenhum relacionamento entre si. Entretanto, como este é um sistema suporte múltiplos pipelines, é possível que existam dois ou mais *pipelines* que repitam os mesmos passos até divergirem em alguma etapa posterior. Eles poderiam até mesmo não divergir em passo algum. Por exemplo, dois *pipelines* podem ter como primeiro passo a execução do módulo *Phred* com exatamente os mesmos parâmetros. Porém, no segundo passo usam parâmetros diferentes para o módulo *Seqclean*. Neste caso, o resultado do primeiro passo poderia ser reaproveitado, já que seria exatamente igual nos dois *pipelines*.

Por isso, é importante que os pipelines sejam armazenados em estruturas de dados onde exista o reaproveitamento do resultado de passos já executados. Ganha-se dessa forma mais eficiência na execução dos *pipelines* e maior economia de espaço em disco. Neste contexto, a estrutura de dados escolhida neste protótipo para se armazenar *pipelines* é uma **árvore**. O conceito da estrutura de dados árvore é descrita na seção 2.6.

A árvore é montada a partir de um nodo raiz neutro, que não está associado a nenhum *pipeline*. Todos os demais nodos armazenam um passo de *pipeline*, e são filhos diretos ou

indiretos deste nodo raiz. A ideia central é que dois *pipelines* inseridos na árvore sigam um mesmo caminho enquanto seus passos forem iguais. No ponto em que os passos divergem, cada um toma um caminho diferente. É importante lembrar que dois passos são iguais se executam exatamente o mesmo módulo com o mesmo conjunto de parâmetros.

A Figura 19 mostra um exemplo das etapas da montagem de uma árvore de *pipelines*. A tabela apresenta três configurações a se inserir na árvore. O primeiro *pipeline* é lido do arquivo de configuração *PipeConf_1*, quando são inseridos três passos P1, P2 e P3 sequencialmente na árvore a partir do nodo raiz. Cada passo possui informações do passo anterior, do módulo a ser executado e dos parâmetros a utilizar.

O segundo *pipeline* é lido do arquivo de configuração *PipeConf_2*. Esse *pipeline* tem um primeiro passo igual ao de *PipeConf_1*, pois ambos executam o mesmo módulo (*phred*) com os mesmos parâmetros (*trim_cutoff=0.16*). Portanto, o primeiro passo de *PipeConf_2* pode reutilizar o resultado do primeiro passo de *PipeConf_1*. O segundo passo de *PipeConf_2* é diferente do segundo nodo de *PipeConf_1*, pois os valores para o atributo *length* não são iguais. O passo P4 é então criado e se torna um dos nodos filhos do nodo P1, o primeiro passo de *PipeConf_1*. O terceiro passo de *PipeConf_2* (P5) é adicionado sequencialmente como nodo filho do último passo inserido (P4).

O terceiro *pipeline* é lido do arquivo *PipeConf_3* e percorre o mesmo caminho que *PipeConf_2* até o segundo passo. A partir do terceiro passo eles divergem (o atributo *coverage* tem valores divergentes), por isso é criada uma bifurcação na árvore neste ponto. É interessante notar que o terceiro passo de PipeConf_3 (P6) executa o mesmo módulo e usa os mesmos parâmetros que o terceiro passo de PipeConf_1 (P3). Porém, como os passos anteriores a eles divergiram, não pode haver um reaproveitamento de resultado, já que cada um acabará recebendo uma entrada de dados diferente.

Para se encontrar os passos de um *pipeline* individual depois da montagem da árvore, basta refazer o caminho inverso, do seu último passo até a raiz. Para se refazer o caminho de *PipeConf_3*, por exemplo, basta navegar na árvore do passo P6 até a raiz. Portanto, após a execução da árvore, o resultado final do *pipeline* criado por *PipeConf_3* será invariavelmente o resultado do passo P6. O mesmo vale para os *pipelines* criados por *PipeConf_1* e *PipeConf_2*, cujos resultados serão iguais aos resultados dos passos P3 e P5, respectivamente.

Config	Passo 1		Passo 2		Passo 3		Legenda
	Módulo	Parâmetros	Módulo	Parâmetros	Módulo	Parâmetros	
PipeConf_1	phred	trim_cutoff = 0.16	seqclean	length = 80 identity = 45	clustering	id_cutoff = 95 coverage = 70	
PipeConf_2	phred	trim_cutoff = 0.16	seqclean	length = 90 identity = 45	clustering	id_cutoff = 95 coverage = 80	
PipeConf_3	phred	trim_cutoff = 0.16	seqclean	length = 90 identity = 45	clustering	id_cutoff = 95 coverage = 70	

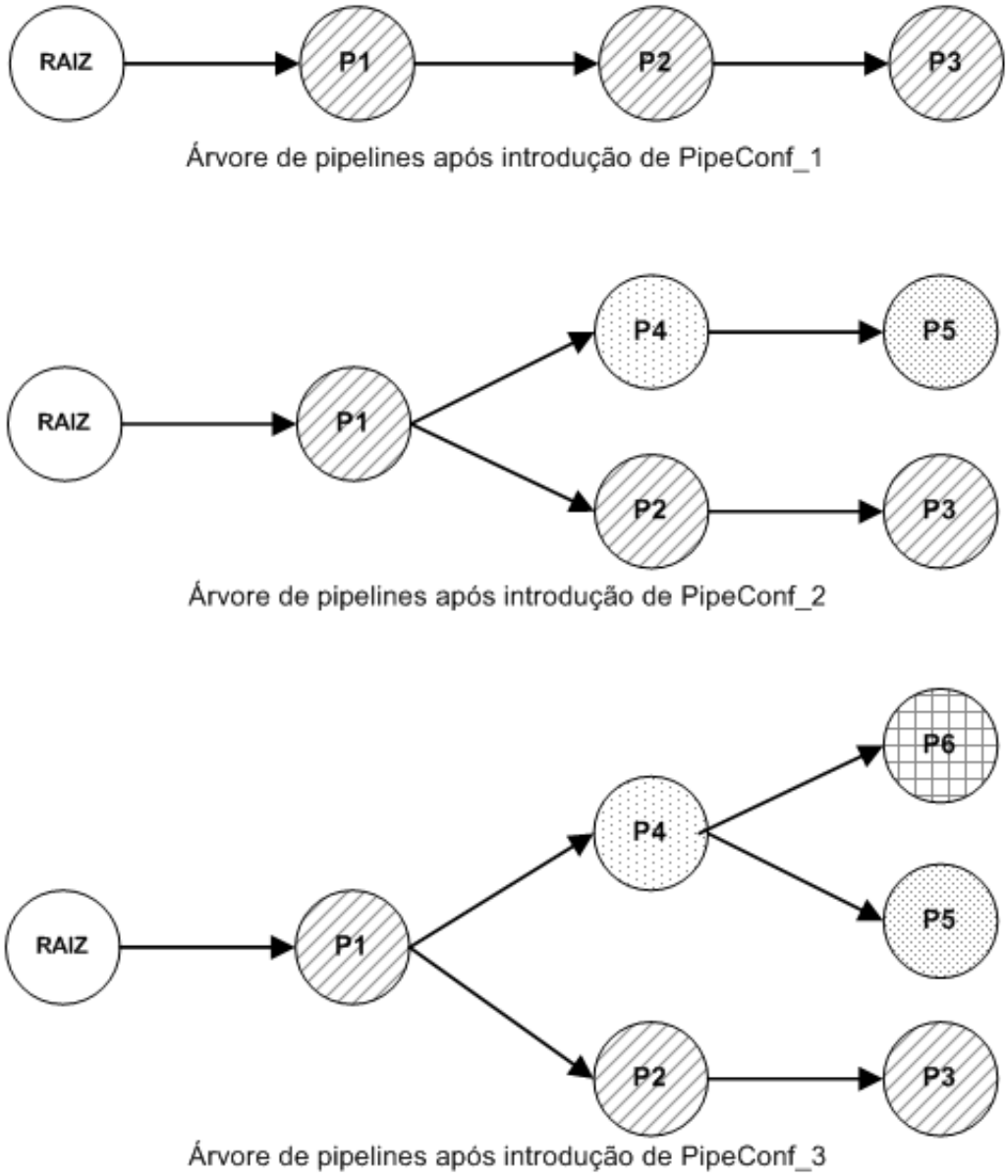

Figura 19 - Exemplo de montagem da árvore de *pipelines*

Esse método de armazenamento de *pipelines* em árvores permite o reaproveitamento do resultado de passos já executados, o que aumenta excepcionalmente a eficiência do

sistema. Quando, por exemplo, novas configurações de *pipeline* são incluídas, uma grande parte dos resultados dos novos *pipelines* gerados já pode estar disponível. Além disso, este tipo de estrutura permite que os passos sejam lidos de forma recursiva e com alto grau de paralelismo.

A Figura 20 mostra o modelo lógico de dados utilizado para armazenamento dos pipelines no banco. As configurações de *pipeline* são armazenadas na tabela *PIPELINE_CONFIG*, as quais estão associadas a um registro da tabela *PIPELINE*. Como foi visto anteriormente, através do último passo é possível obter todos os outros passos do *pipeline*, uma vez que a árvore tem natureza recursiva. Desse modo, cada *pipeline* armazenado na tabela *PIPELINE* possui um atributo nomeado *endStepId*¸ que identifica seu último passo. Os passos de *pipeline* estão armazenados na tabela *PIPELINE_STEP* a qual se relaciona com ela própria através do atributo *parentStepId*. Esse atributo identifica o passo pai de um determinado passo e propicia recursividade à arvore. A tabela *MODULO_PARAM* armazena os parâmetros de cada passo.

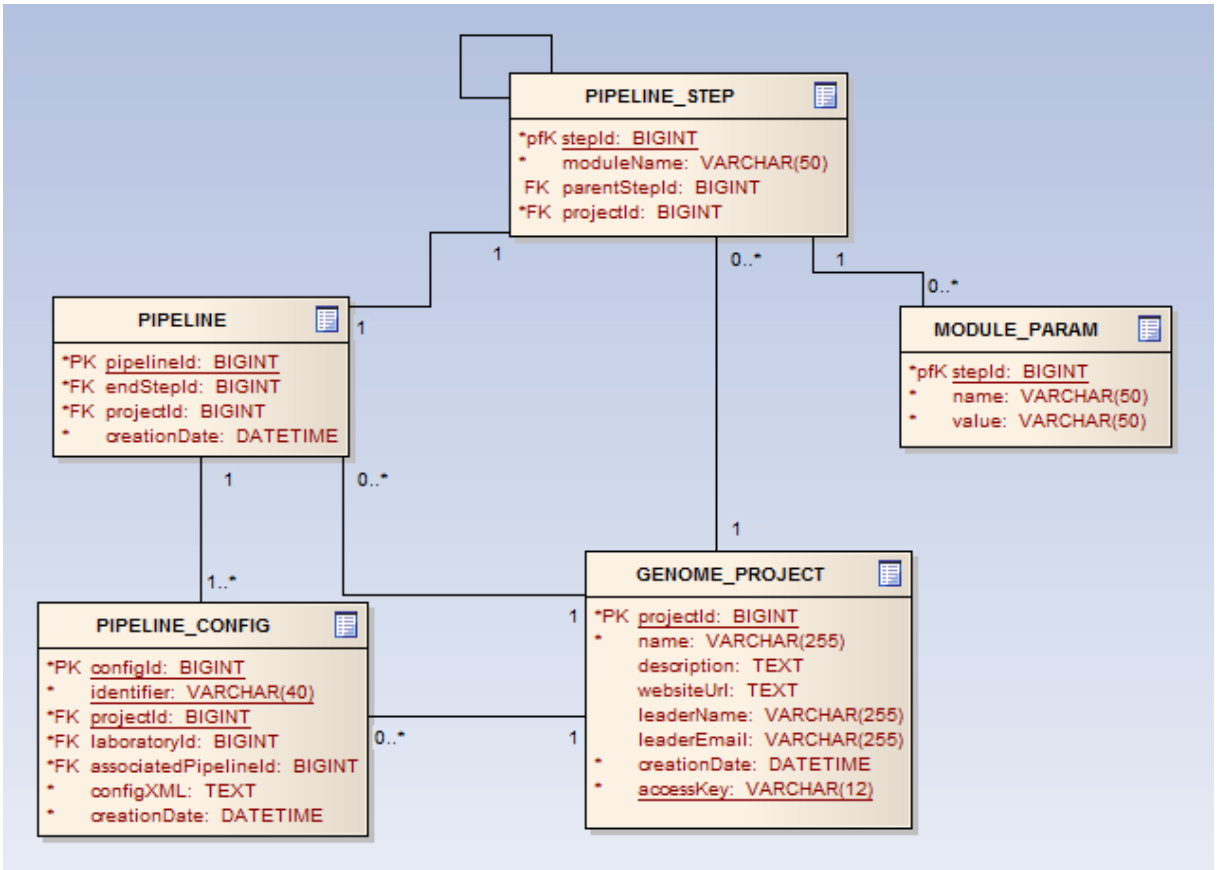

Figura 20 - Modelo lógico de dados para armazenamento de *pipelines*

5.7 Submissão de Cromatogramas

Depois que um projeto foi configurado e *pipelines* foram criados, o pesquisador inicia a submissão de cromatogramas. Cada submissão corresponde a uma placa sequenciada.

O componente *Web* do protótipo disponibiliza um formulário para a submissão de cromatogramas, mostrado na Figura 21.

Como é possível observar na Figura 21, o pesquisador deve preencher os seguintes campos para ter sucesso no envio de cromatogramas:

- Laboratório: deve-se escolher o laboratório que fez o sequenciamento da placa.
- Biblioteca: deve-se escolher a biblioteca genômica a qual pertence a placa sequenciada.
- Placa: deve-se informar o número da placa, com três dígitos.
- Direção: deve-se informar a direção do sequenciamento (5' ou 3').
- Tentativa: deve-se informar o número da tentativa de sequenciamento da placa (no caso de falhas).
- Arquivo *zip*: deve-se selecionar o arquivo *zip* contendo os cromatogramas gerados a partir da placa.

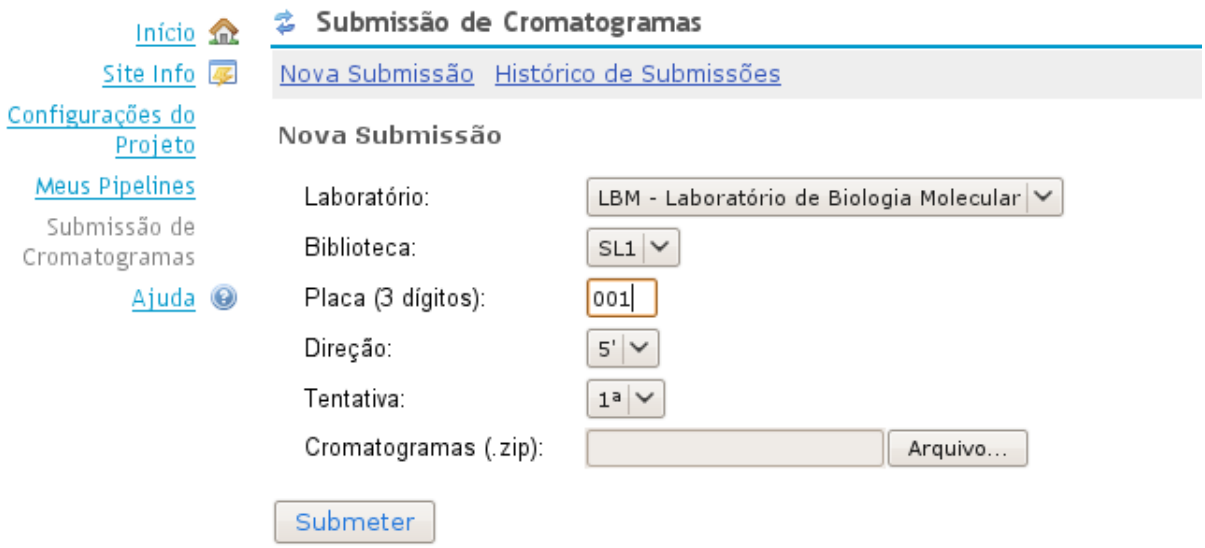

Figura 21 - Formulário para submissão de cromatogramas

Todos os cromatogramas de uma placa sequenciada devem estar em um arquivo compactado no formato *ZIP*, e nomeados de acordo com a posição na placa (ex: A01.esd,

A02.esd, D03.esd, etc). Os formatos aceitos são aqueles compatíveis com o programa *Phred*: ESD, ABI e SCF [17].

Após a submissão do formulário, o componente *Web* chama o método *submitChromat* do *Web Service*, passando os valores dos campos do formulário, além da chave de acesso ao projeto. O *Web Service*, por sua vez, chama a API lógica do Núcleo, que armazena uma nova submissão no banco de dados.

A Figura 22 mostra o modelo lógico de dados utilizado para armazenamento de submissões de cromatograma. A tabela CHROMAT armazena cromatogramas individuais, cada qual associado a um registro da tabela *SUBMISSION*. Essa última armazena as submissões individuais, que correspondem a uma placa sequenciada. Cada registro nessa tabela está associado a um projeto, um laboratório e uma biblioteca genômica. A tabela *PROCESSED_SUBMISSION* guarda as submissões que já foram processadas por um determinado passo de *pipeline.* Se um erro ocorre durante o processamento de uma submissão, ela é marcada como inválida.

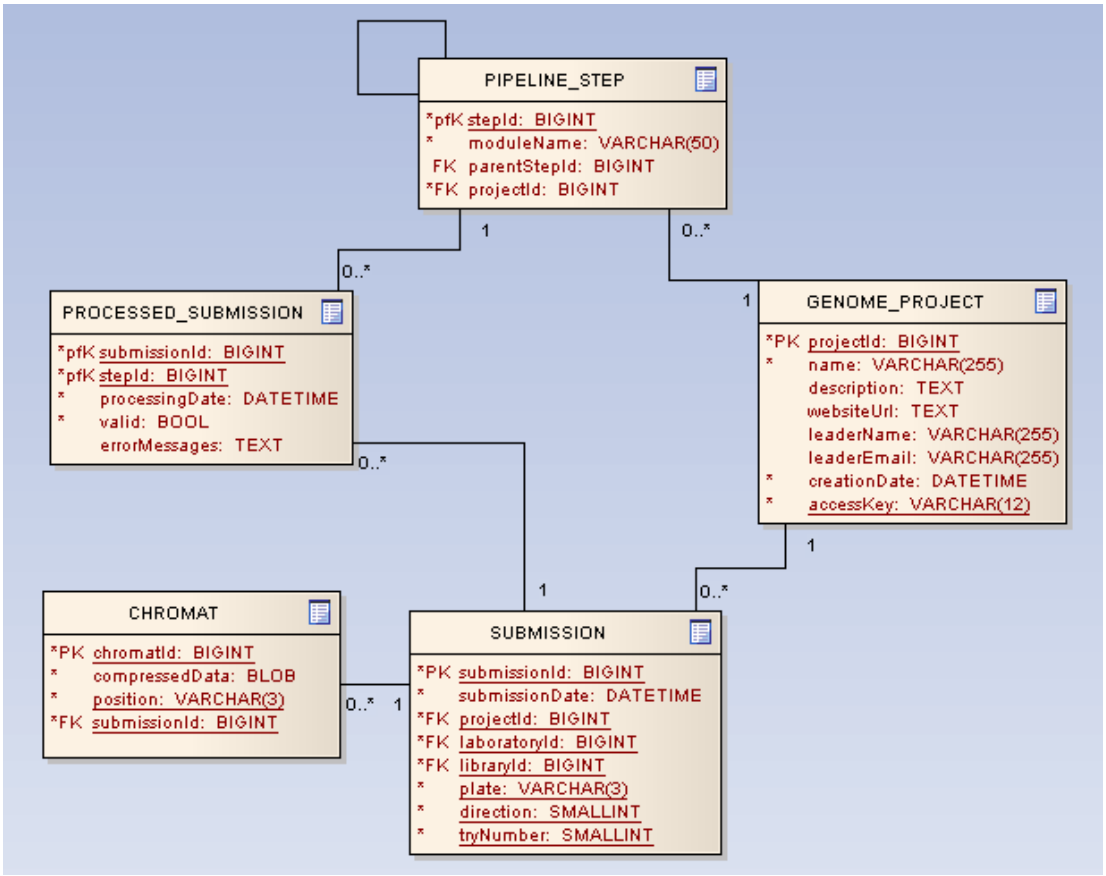

Figura 22 - Modelo lógico de dados para armazenamento de submissões de cromatogramas

5.8 Execução de *Pipelines*

A execução de *pipelines* é o processo em que todos os *pipelines* de um projeto são disparados. Durante esse processo, os *pipelines* fazem a leitura e o processamento de todas as submissões de cromatogramas recebidas e ainda não processadas.

A execução de *pipelines* ocorre periodicamente no componente Núcleo, por meio da ferramenta *cron* [47]. Essa ferramenta é capaz de agendar a execução de *jobs.* A periodicidade de execução deve ser configurada de acordo com a frequência em que submissões de cromatogramas são recebidas.

O processo de execução de *pipelines* consiste na travessia da árvore de *pipelines*. Durante a travessia, os módulos associados a cada passo são executados com os devidos parâmetros.

5.8.1 Travessia da Árvore de Pipelines

A travessia da árvore de *pipelines* é realizda em um processo conhecido como pré-ordem. A travessia pré-ordem pode ser definida recursivamente, através do seguinte algoritmo:

```
Preordem(umNodo):
  Processa umNodo
  Para cada filho de umNodo (se houver)
     Executa Preordem(filho)
```

Dessa forma, um nodo da árvore processa o seu código, e em seguida chama a execução de cada um de seus nodos filhos. Os nodos filhos, por sua vez, farão o mesmo processo. Este processo se inicia no nodo raiz da árvore e termina nos nodos correspondentes às folhas.

A travessia pré-ordem é particularmente útil para o caso de árvores de *pipelines*. Isso porque um determinado passo B depende da saída de dados do passo anterior A. Desse modo, o passo B só pode ser executado depois do passo A, o que é exatamente o que ocorre no algoritmo de travessia pré-ordem. Para o caso da árvore de *pipelines*, um novo recurso é adicionado para que a travessia se torne mais eficiente. Quando ocorre uma bifurcação na árvore, os dois ou mais caminhos que são frutos dessa bifurcação se tornam independentes entre si, e por isso podem ser executados em paralelo. Desse modo, o recurso de paralelismo

pode ser adicionado à travessia pré-ordem. Sempre que ocorre uma bifurcação na árvore, as novas subárvores que surgem a partir dela são executadas em *threads* separadas. Este recurso aumenta consideravelmente a eficiência da execução dos *pipelines*, principalmente quando o sistema é executado em plataformas multiprocessadas.

O algoritmo pré-ordem adaptado à árvore de *pipelines* é definido, portanto, da seguinte forma:

```
Preordem(umNodo):
  Processa umNodo
  Para cada filho de umNodo (se houver)
     Executa Preordem(filho) em thread separada
```

A Figura 23 ilustra o processo de travessia da árvore de *pipelines*. No primeiro quadro, o primeiro passo é executado. O sistema aguarda o fim da execução do módulo relacionado ao primeiro passo e então dispara duas *threads* executando os dois passos filhos de forma paralela, como pode ser visto no segundo quadro. Quando estes últimos terminam a execução, seus filhos também são executados em paralelo, como demonstra o quadro 3.

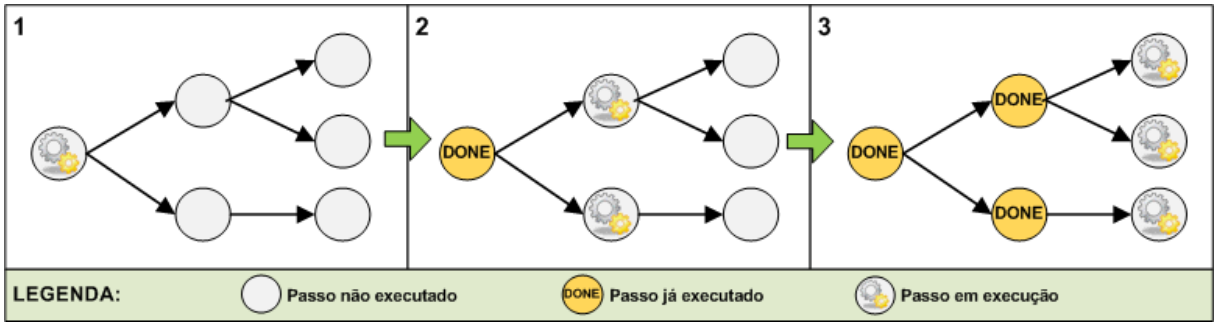

Figura 23 - Travessia da árvore de *pipelines*

5.8.2 Pool de threads

O modelo descrito na seção 5.8.1 é eficiente, porém, com o crescimento da árvore pode-se chegar a um ponto em que existe um número excessivo de threads em execução simultânea. Esse é um fator que pode reduzir o desempenho do sistema, pois existe um *overhead* associado à criação e destruição de threads.

A estratégia utilizada para se evitar um número muito grande de *threads* em execução é o uso do padrão de projetos *pool de threads*. Esse padrão define uma coleção de *threads* disponível para a realização de tarefas. Dessa forma, se estabelece um número máximo de

threads para a coleção. As tarefas que necessitam de execução aguardam em uma fila até que exista uma *thread* disponível para executá-la.

No caso da árvore de *pipelines*, a execução de cada nodo torna-se uma tarefa que entra na fila de execução do *pool de threads*. O número máximo de *threads* disponível no *pool* deve ser um atributo configurável. Essa é uma forma de se otimizar o desempenho do sistema em função do tamanho médio da árvore de *pipelines*.

5.9 Módulos

No sistema proposto, um módulo é um componente que encapsula a lógica de um programa ou algoritmo usado para a execução de um passo de *pipeline*.

Conforme visto anteriormente, um passo de *pipeline* contém informações sobre o módulo a executar e também os parâmetros a se utilizar na execução do mesmo. Portanto, o módulo é a unidade que de fato executa e gera resultados para um determinado passo.

Cada módulo executa transformações nos dados disponíveis a partir de um conjunto de parâmetros. Em uma visão geral, cada módulo obtém dados do passo anterior, processa estes dados e os disponibiliza para o próximo passo. A Figura 24 ilustra o comportamento típico de um módulo.

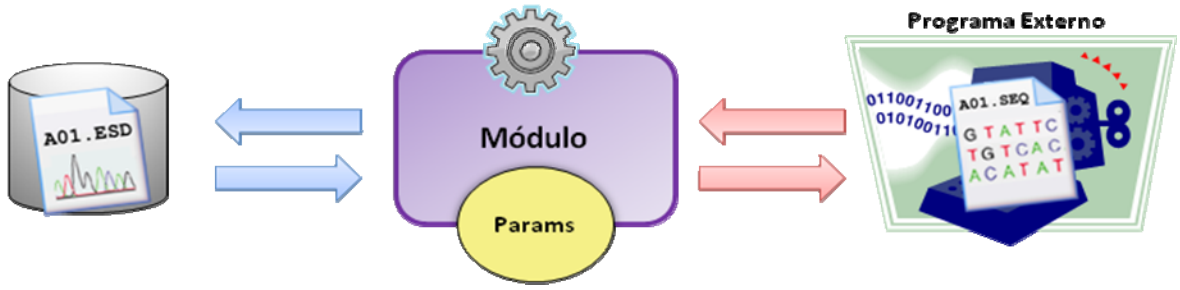

Figura 24 - Comportamento típico de um módulo

Neste protótipo, a comunicação entre os passos é toda feita via banco de dados. Um módulo obtém seus dados de entrada em tabelas específicas e grava os resultados de seu processamento novamente no banco. O módulo do próximo passo efetua o mesmo processo, utilizando os dados gerados pelo seu antecessor. Neste modelo, não existe comunicação direta entre dois módulos, o que faz com que cada módulo seja um componente independente. Entretanto, uma desvantagem deste sistema é que o módulo deve "conhecer" todas as tabelas do banco, deixando o sistema de certa forma acoplado ao modelo de dados.

Em *pipelines* de análise genômica, geralmente cada passo é executado por um programa externo, tais como *phred, seqclean, cap3* e outros. Os módulos, nestes casos, deverão encapsular a lógica de acesso ao sistema operacional para chamar os programas, assim como a tradução e inserção dos resultados da execução nas tabelas do banco de dados da aplicação. Existe a possibilidade, porém, da criação de módulos com algoritmos próprios, sem o uso de programas externos.

Com o intuito de prover extensibilidade de código, a implementação de módulos é independente da lógica do sistema. A ligação entre a implementação e o sistema é feita de forma declarativa, em documento XML. No entanto, para que um módulo seja compatível com o sistema, o mesmo deve seguir o contrato da interface *PipeModule*. O contrato dessa interface exige que as implementações sobrescrevam dois métodos: *executeModule* e *validateParams*.

O método *executeModule* é invocado por um passo de pipeline de modo a realizar a execução do passo. Este método deve executar a lógica do módulo, isto é, fazer as transformações necessárias nos dados a fim de cumprir o seu objetivo no processo de análise. O método *validateParams* é chamado no momento em que um novo *pipeline* será inserido no sistema, a fim de se verificar se os parâmetros passados em cada módulo são válidos.

A Figura 25 ilustra a interface *PipeModule* e possíveis implementações.

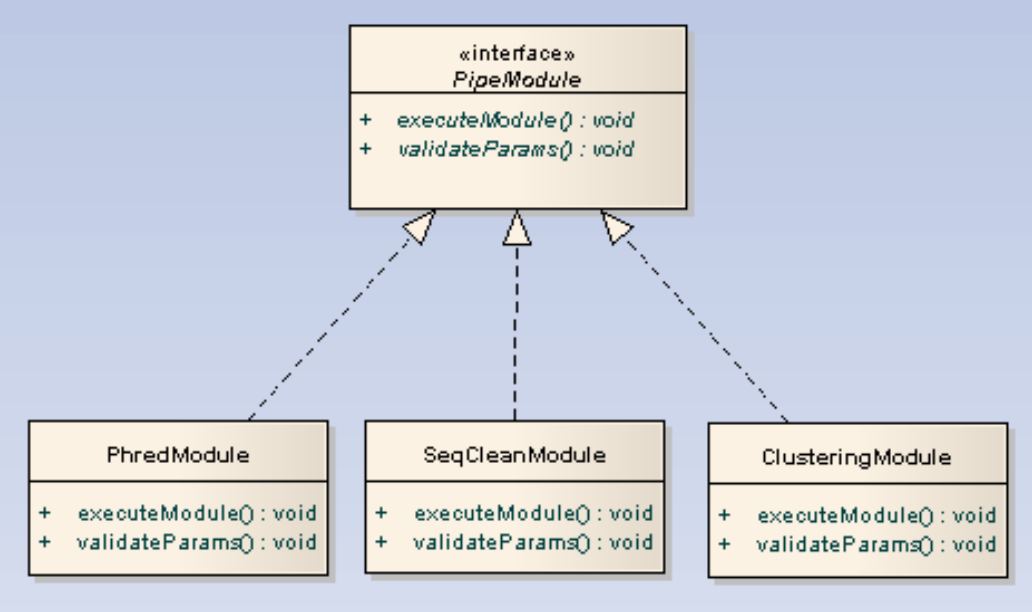

Figura 25 - Interface *PipeModule* e possíveis implementações

A Figura 25 ilustra um exemplo em que existem três implementações da interface *PipeModule: PhredModule, SeqCleanModule* e *ClusteringModule*. Os dois primeiros concretizam a interface encapsulando as lógicas dos programas *phred* e *seqclean*, enquanto o último implementa a etapa de agrupamento de sequências.

O modelo apresentado permite que o sistema esteja fortemente desacoplado de implementações específicas de módulos. Este fato aumenta a flexibilidade e a extensibilidade da aplicação, uma vez que diferentes implementações podem ser usadas e novos módulos podem ser integrados sem a necessidade de recompilação.

No protótipo desenvolvido neste trabalho foram desenvolvidos cinco módulos que implementam a interface *PipeModule*. No capítulo 6, o relacionamento entre estes módulos e a lógica de cada um deles são apresentados com maiores detalhes.

5.10 Visualização de Resultados

A visualização de resultados das análises é realizada no componente *Web*. O pesquisador deve primeiro escolher o *pipeline* que deseja visualizar resultados. Em seguida deve escolher entre os relatórios disponíveis.

O pesquisador seleciona um dos relatórios e o componente *Web* faz uma chamada à operação correspondente no *Web Service*. O *Web Service*, por sua vez, possui lógica para obtenção dos dados necessários para a montagem dos relatórios. Para isso, faz chamadas ao componente Núcleo. O *Web Service* retorna um objeto contendo os dados do relatório gerado para o componente Web, que o exibe com a formatação adequada.

5.11 Considerações Finais

Nas sessões anteriores, foi apresentado o protótipo de um sistema de análise genômica baseado em *Web services*. O sistema provê ainda suporte a múltiplos *pipelines* e acompanhamento progressivo de projetos.

O uso de *Web services* desacopla a lógica do sistema de interfaces gráficas específicas. Um mesmo projeto pode ser manipulado através de sistemas de naturezas diferentes, como aplicações *Web* heterogêneas ou outras ferramentas.

O modelo de armazenamento de *pipelines* em uma árvore tem vantagens significativas, tais como o reaproveitamento dos resultados de passos já executados e o processamento paralelo de subárvores independentes. Entretanto, tem também algumas desvantagens, tal como a necessidade de se armazenar o resultado de todos os passos.

Por fim, o modelo apresentado para integração de módulos ao sistema também o deixa bastante flexível. O sistema é totalmente desacoplado de implementações específicas de módulos. Essas são integradas ao sistema com simples configurações em arquivos XML. Dessa forma, novos módulos podem ser facilmente incluídos.

6. MÓDULOS

6.1 Considerações Iniciais

No protótipo construído neste trabalho e descrito no capítulo 5, os módulos encapsulam a lógica de um passo de *pipeline*. Cada implementação de módulo é responsável por uma etapa do processo de análise genômica.

Cada passo de um *pipeline* está associado a um módulo e a um conjunto de parâmetros para esse módulo. Quando um *pipeline* é executado, seus passos são processados individualmente, um após o outro. O processamento de um passo se dá pela execução do módulo ao qual está associado, utilizando o conjunto de parâmetros definido para o passo.

Cinco módulos foram implementados para o protótipo desenvolvido, sendo que cada um deles é responsável por uma etapa no processo de análise genômica.

Os módulos implementados são: *Phred*, responsável pela etapa de *base calling* (reconhecimento das bases); *SeqClean*, responsável pela etapa de limpeza das sequências de baixa qualidade; *Clustering*, responsável pelo agrupamento de sequências em *clusters*; *CAP3*, responsável pela formação de *contigs*; e *Pathways*, responsável por descobrir funções e vias metabólicas associadas aos *contigs* encontrados. Como é possível observar, a maioria dos módulos foi nomeada de acordo com o principal programa externo que executa.

A Figura 26 mostra relacionamento entre estes módulos.

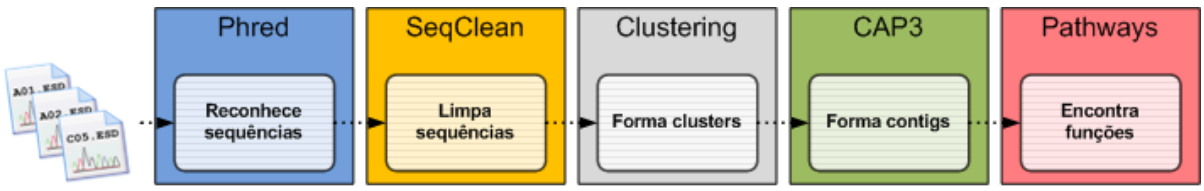

Figura 26 - Relacionamento entre módulos

O funcionamento de cada desses módulos será explicado com detalhes nas próximas seções.

6.2 Phred

O Módulo *Phred* é responsável pela etapa de *base calling* de um pipeline. Nessa etapa, a natureza e a ordem das bases presentes em um cromatograma são determinadas (ver seção

3.3.1 para maiores detalhes sobre essa etapa). O resultado da execução da etapa é o registro de uma sequência no banco de dados para cada cromatograma válido.

Este módulo essencialmente é um adaptador para o programa externo *phred* [17]. O módulo lê como entrada o conjunto de submissões de cromatogramas ainda não processadas pelo passo corrente, executa *phred* e então armazena os arquivos de sequências e qualidade gerados por *phred* no banco de dados.

O único parâmetro que o módulo aceita é *trim_cutoff*. Esse parâmetro é utilizado na execução de *phred* e define a porcentagem de erro para que uma base seja considerada de baixa qualidade. Regiões de baixa qualidade são eliminadas na etapa de limpeza do *pipeline*. Por isso, a variação do valor desse parâmetro pode ter grande impacto nos resultados gerados pela análise.

A Figura 27 mostra os passo-a-passo de execução do módulo.

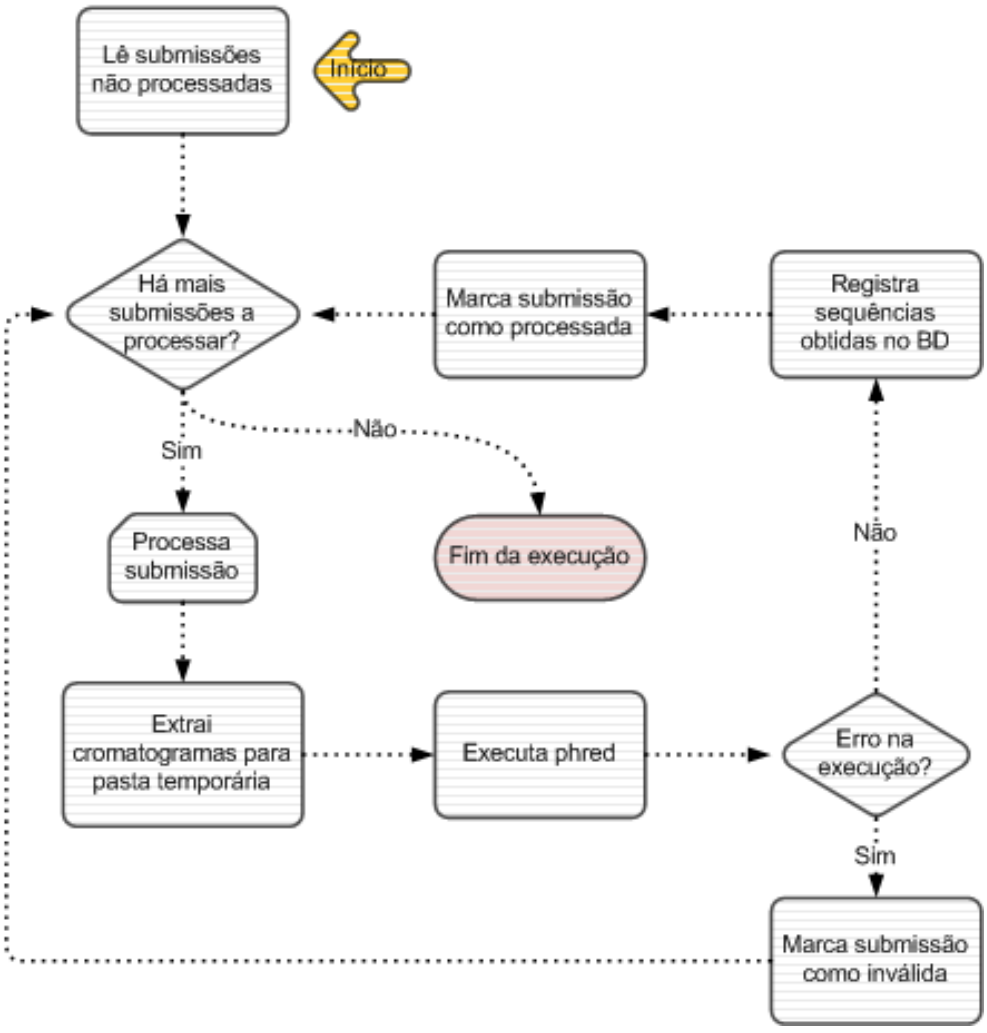

Figura 27 – Passo-a-passo de execução do módulo Phred

No início de sua execução, o módulo lê no banco de dados todas as submissões de cromatogramas que não foram processadas pelo passo de *pipeline* corrente. Cada submissão corresponde a uma placa sequenciada e possui um arquivo *zip* que armazena os cromatogramas gerados a partir da placa.

Para cada submissão obtida, os cromatogramas do arquivo *zip* são extraídos para uma pasta temporária, onde então o programa *phred* é executado. A linha de comando utilizada para execução de phred é a seguinte:

```
phred -id chromat_dir -trim_alt "" -trim_cutoff <param_trim_cutoff>
```

Nesta linha, <*param_trim_cutoff*> é substituído pelo parâmetro de módulo *trim_cutoff*.

Os parâmetros *trim_cutoff* e *trim_alt* são inter-relacionados, uma vez que esse depende do valor daquele. *Trim_alt* zera no arquivo de qualidade gerado todas as bases com valor de qualidade baixo. *Trim_cutoff* determina a porcentagem de erro para que uma base seja considerada de baixa qualidade.

A execução de *phred* gera no diretório temporário um arquivo FASTA contendo as sequências de bases lidas dos cromatogramas e um arquivo *.qual* contendo os valores de qualidade associados a cada base das sequências obtidas. O próximo passo na execução do módulo é ler esses arquivos e criar um registro no banco de dados para cada sequência lida, contendo também seus valores de qualidade. Outros dados importantes também são registrados, como o tamanho da sequência e as posições onde iniciam e terminam as bases de boa qualidade (determinadas pelo parâmetro *trim_cutoff*).

Caso não ocorra nenhum erro no processamento de *phred*, a submissão é registrada como processada para este passo. Submissões que possuam cromatogramas defeituosos são registradas como inválidas.

A Figura 28 mostra o modelo lógico de dados utilizado pelo módulo *Phred*. Os arquivos *zip* contendo os cromatogramas de cada submissão são obtidos do campo *compressedData* da tabela *CHROMAT*. Depois do processamento de *phred*, o módulo gera registros na tabela *SEQUENCE* contendo a sequência de bases e os valores de qualidade para cada cromatograma. A tabela *GENERATED_SEQUENCE* informa as sequências válidas que foram geradas em um determinado passo.

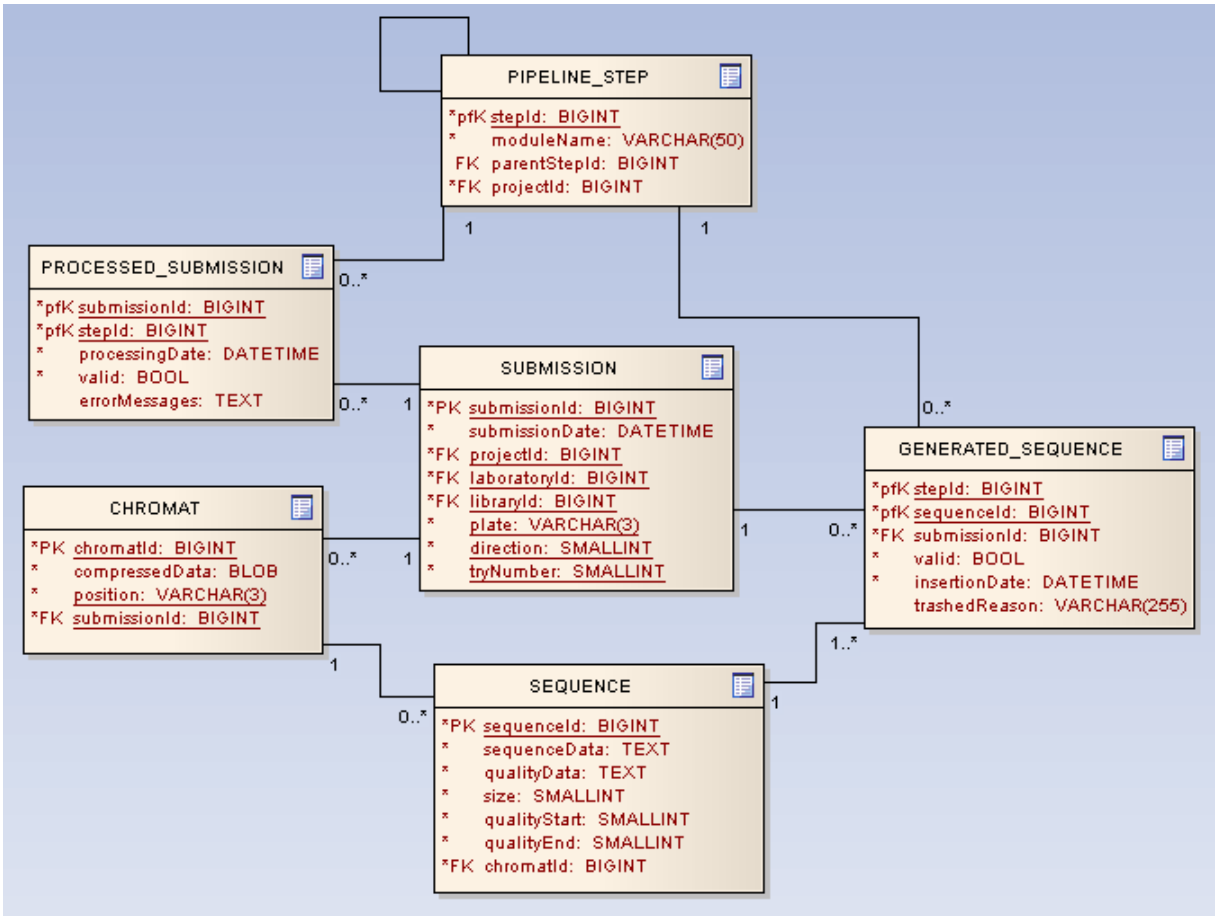

Figura 28 - Modelo lógico de dados utilizado pelo módulo Phred

6.3 SeqClean

A segunda etapa do pipeline construído neste trabalho é executada pelo módulo *SeqClean*, o qual é um adaptador para o programa externo de mesmo nome.

Nessa etapa, as sequências geradas no processo anterior passam por um processo de limpeza. As sequências consideradas de baixa qualidade são descartadas, pois podem levar a resultados imprecisos ou com grande margem de erro nos passos seguintes.

O programa s*eqclean* faz a limpeza das sequências em dois passos. No primeiro passo o programa corta segmentos no início ou no final das mesmas, de acordo com os seguintes critérios:

- Porcentagem de bases indeterminadas (marcadas como “N” por *phred*).
- Presença de cauda poli-A.
- Presença de sequências de vetores, *primers* e adaptadores utilizados no processo de sequenciamento.

- Presença de outros contaminantes (DNA de espécies diferentes do organismo estudado).

Ao final do primeiro passo, o tamanho das sequências é reduzido devido aos cortes. As sequências reduzidas são utilizadas no segundo passo, no qual ocorre o descarte daquelas que se enquadrem em pelo menos um destes dois casos: tamanho da sequência abaixo de um limite pré-determinado ou porcentagem de bases indeterminadas acima de 3% do total de bases da sequência.

O módulo *SeqClean* permite a configuração de dois parâmetros. O parâmetro *min_length* determina o limite mínimo de tamanho para que uma sequência não seja descartada pelo programa *seqclean*. O valor deste parâmetro é usado na opção "-l" do programa *seqclean*.

O parâmetro *min_perc_identity* define o percentual mínimo de identidade no alinhamento com contaminantes. O valor deste parâmetro é usado na opção "-x" de *seqclean*.

A Figura 29 mostra o passo-a-passo do módulo *SeqClean*.

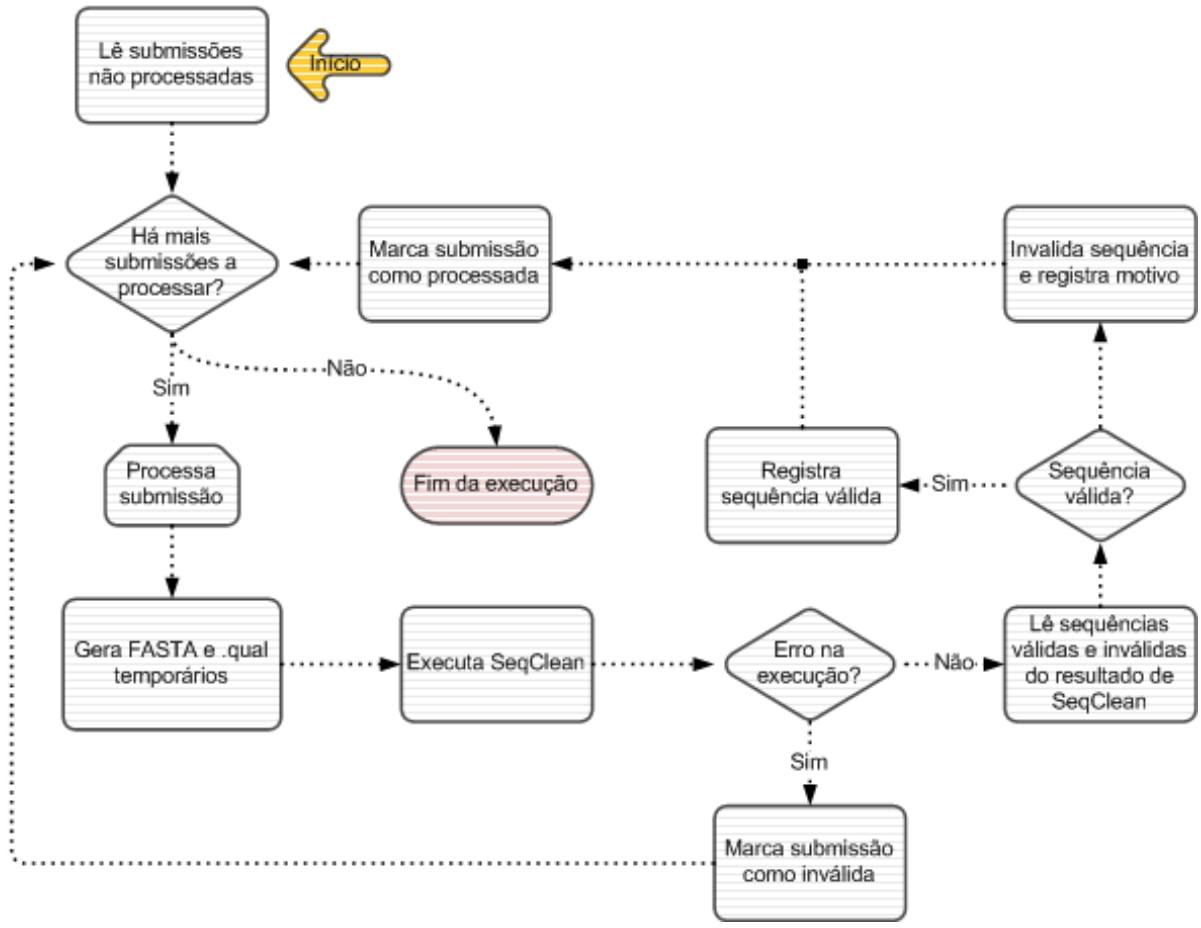

Figura 29 - Passo-a-passo de execução do módulo *SeqClean*

Inicialmente o módulo obtém as submissões ainda não processadas pelo passo corrente (o processamento é sempre feito submissão a submissão). Para cada submissão A, obtém então o conjunto de sequências associadas a A, geradas na etapa anterior do pipeline (pelo módulo *Phred*).

Um arquivo FASTA contendo todas as sequências e outro contendo os valores de qualidade para cada uma delas são gerados em um diretório temporário. O programa *seqclean* é então executado. A seguinte linha de comando é utilizada para execução de *SeqClean*:

```
seqclean join.fasta -l <param_min_length> -x <param_min_perc_identity> -v
<contaminant_dbs>
```

Nessa linha, "join.fasta" representa o arquivo FASTA de entrada, contendo as sequências que passarão pelo processo de limpeza. <*param_min_lenght*> e <*param_min_perc_identity*> são substituídos na linha pelos valores dos parâmetros de módulo *min_length* e *min_perc_identity*, respectivamente.

O parâmetro -v do programa *SeqClean* aceita uma lista de bases de dados de contaminantes. Esta lista é usada para se identificar e limpar possíveis contaminantes, vetores e adaptadores presentes nas sequências. O valor desta opção na linha de comando depende da submissão que está sendo processada. Uma submissão está associada a uma biblioteca genômica, que por sua vez, está associada a um conjunto de contaminantes. Portanto, a opção -v recebe uma lista de caminhos para as bases de dados de contaminantes registrados para a biblioteca da submissão corrente.

Após a execução, *SeqClean* gera dois arquivos, com extensões *.clean* e *.cln*. O arquivo *.clean* contém as sequências válidas após a limpeza. As sequências contidas neste arquivo já estão livres de contaminantes e áreas de baixa qualidade. Cada uma delas é armazenada no banco de dados para que possam ser utilizadas na próxima etapa do *pipeline*.

O arquivo *.cln* mostra as sequências que foram descartadas e as causas do descarte. Estas sequências são marcadas como inválidas no banco de dados e o motivo do descarte também é armazenado. Sequências invalidadas não serão utilizadas na próxima etapa do *pipeline*.

Caso não ocorram erros no processamento de *seqclean,* a submissão é registrada como processada para o passo corrente.

O modelo lógico de dados utilizado pelo módulo *SeqClean* é o mesmo que o utilizado pelo módulo *Phred*, apresentado na Figura 28. O módulo SeqClean também gera registros na tabela *GENERATED_SEQUENCE* para todas as sequências analisadas. A diferença entre os

dois é que *SeqClean* preenche os campos *valid* e *trashed_reason* de sequências que foram invalidadas. O primeiro recebe o valor *falso,* indicando que a sequência é inválida. O segundo recebe o motivo do descarte, indicado pelo programa *SeqClean.*

6.4 Clustering

A próxima etapa no *pipeline* é o processo de agrupamento de sequências. O objetivo do agrupamento é criar grupos (ou *clusters*) de sequências com alto grau de identidade entre si, isto é, aquelas que possuem bases iguais em várias posições.

A verificação do grau de alinhamento entre as sequências é feita pelo programa externo *MegaBLAST*. Esse programa tem duas entradas: um arquivo FASTA e uma base de dados formatada, ambos formados por sequências de nucleotídeos. Na linguagem de *MegaBLAST*, o arquivo FASTA é denominado *query* e a base é chamada de *subject*. O programa forma pares de sequências *query-subject* com alto grau de identidade.

Os pares de sequências formados pelo programa *MegaBLAST* são utilizados pelo algoritmo de agrupamento do módulo, denominado *Single Linkage*. Este algoritmo coloca em um mesmo *cluster* todas as sequências que formam pelo menos um par (isto é, possui alto grau de identidade) com alguma das sequências já existentes no *cluster*. Por exemplo, se as sequências A e B formam um par, e as sequências B e C formam outro par, então A, B, C pertencem a um mesmo *cluster*.

O módulo *Clustering* permite a configuração de dois parâmetros. O parâmetro *identity_cutoff* define o percentual de identidade entre duas sequências para que as mesmas pertençam a um mesmo *cluster*. O parâmetro *min_length_coverage* determina a cobertura que uma sequência deve ter em relação a outra para que as duas formem um par. A cobertura será explicada mais adiante neste texto.

A Figura 30 mostra o passo-a-passo de execução do módulo *Clustering*.

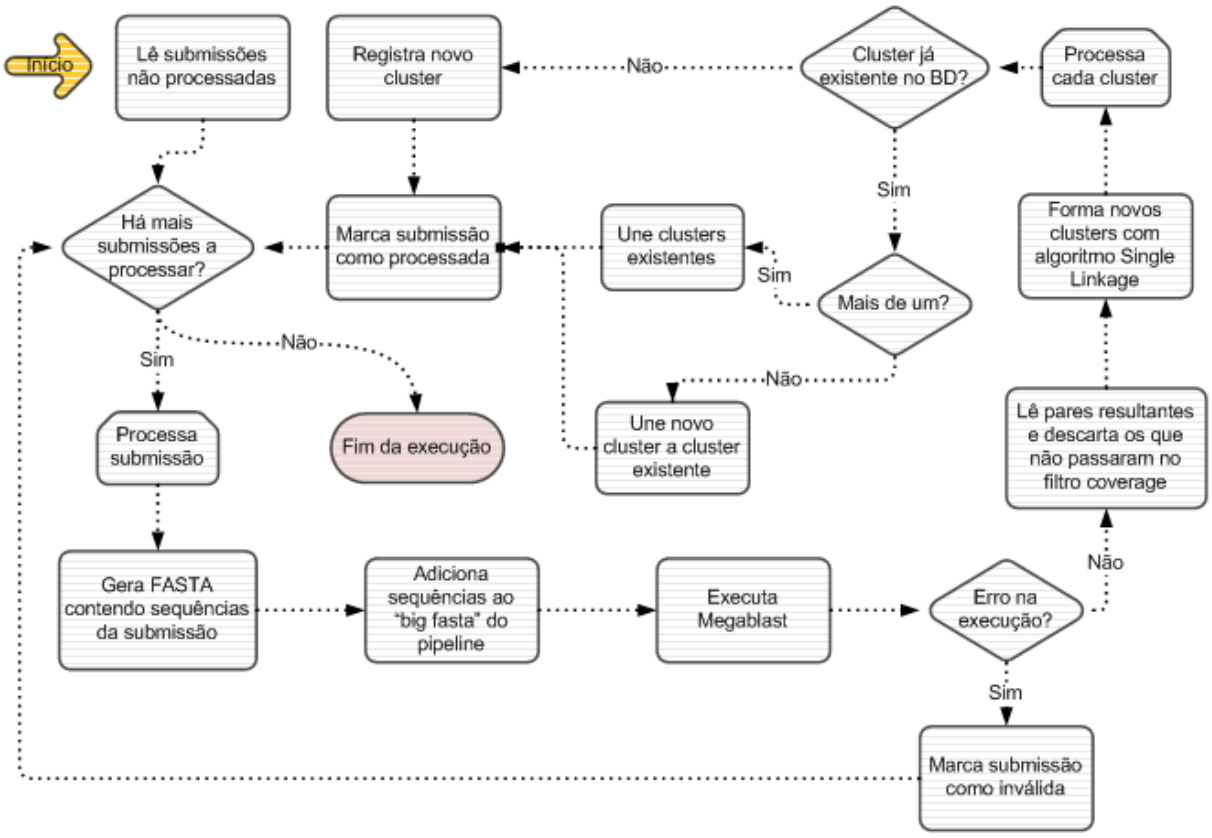

Figura 30 - Passo-a-passo de execução do módulo Clustering

O módulo obtém primeiramente as submissões não processadas para o passo corrente. Para cada uma das submissões obtidas, obtém as sequências geradas pelo passo anterior, isto é, as sequências livres de contaminantes que não foram descartadas pelos filtros do módulo *SeqClean*.

A formação de *clusters* pode ocorrer tanto entre as próprias sequências obtidas como entre aquelas obtidas e as outras armazenadas no banco (geradas em outras submissões). Desse modo, a *query* usada no programa *MegaBLAST* é um arquivo FASTA contendo as sequências obtidas da submissão corrente, enquanto o *subject* é formado por essas sequências mais todas aquelas geradas em submissões anteriores. No sentido de evitar que a cada execução o módulo tenha que gerar novamente o arquivo FASTA contendo todas as sequências do projeto, este arquivo é mantido em um diretório interno do sistema após ser gerado pela primeira vez. Um grande arquivo FASTA denominado *big fasta* é mantido para cada *pipeline* registrado no sistema, sendo que a cada execução do módulo *Clustering*, novas sequências são adicionadas a esse arquivo.

A linha de comando usada no programa *MegaBLAST* é a seguinte:

```
megablast -i <query> -d <subject> -o megablast.result -p <param_identity_cutoff> -m
8 -F F
```

Nessa linha, <*query*> e <*subject*> são substituídos respectivamente pelo caminho para o arquivo FASTA com as novas sequências e pelo caminho para a base de dados contendo todas as sequências do projeto. <*param_identity_cutoff*> define o percentual de identidade mínimo para que duas sequências formem um par. O parâmetro -m 8 define o tipo de saída tabular, o qual facilita a leitura dos resultados. O parâmetro -F F desliga todos os filtros, uma vez que a filtragem já foi efetuada por *SeqClean* na etapa anterior do *pipeline*.

O arquivo de saída gerado pelo programa *MegaBLAST* lista os pares de sequências com identidade maior que *identity_cutoff*. O módulo lê os pares formados um a um, descartando aqueles que não passam no filtro de *coverage*. Esse filtro calcula a porcentagem de cobertura do alinhamento resultante em relação ao tamanho total possível de alinhamento entre as duas sequências. Se esse percentual ficar abaixo daquele determinado pelo parâmetro de módulo *min_length_coverage*, o par é descartado, não podendo formar *clusters*.

A partir da leitura dos pares não descartados pelo filtro, o módulo então usa o algoritmo *Single Linkage* para formar *clusters* de sequências. Após a execução do algoritmo, se, por exemplo, uma sequência A foi colocada em um *cluster* C, existe pelo menos mais uma sequência no cluster C que forma um par com a sequência A.

Depois que os clusters são formados, é necessário ainda verificar se é possível uni-los a *clusters* já existentes no banco de dados. Para isto, verifica-se em cada um dos novos *clusters* se há sequências que pertencem a *clusters* já existentes. Caso existam, o novo *cluster* se une ao *cluster* já existente. É interessante notar que dois ou mais *clusters* já existentes podem também se unir neste processo. Isso acontece quando em um mesmo novo *cluster* existem sequências que pertencem a mais de um *cluster* já existente.

Um novo *cluster* é criado no banco quando não são encontradas sequências pertencentes a *clusters* já existentes. Os novos *clusters* e os *clusters* alterados pelo passo corrente são marcados como "atualizados", para que a próxima etapa do *pipeline* os identifique.

Caso não ocorram erros no processamento do módulo, a submissão é registrada como processada para o passo corrente.

A Figura 31 mostra o modelo lógico de dados usado para armazenamento de *clusters*.

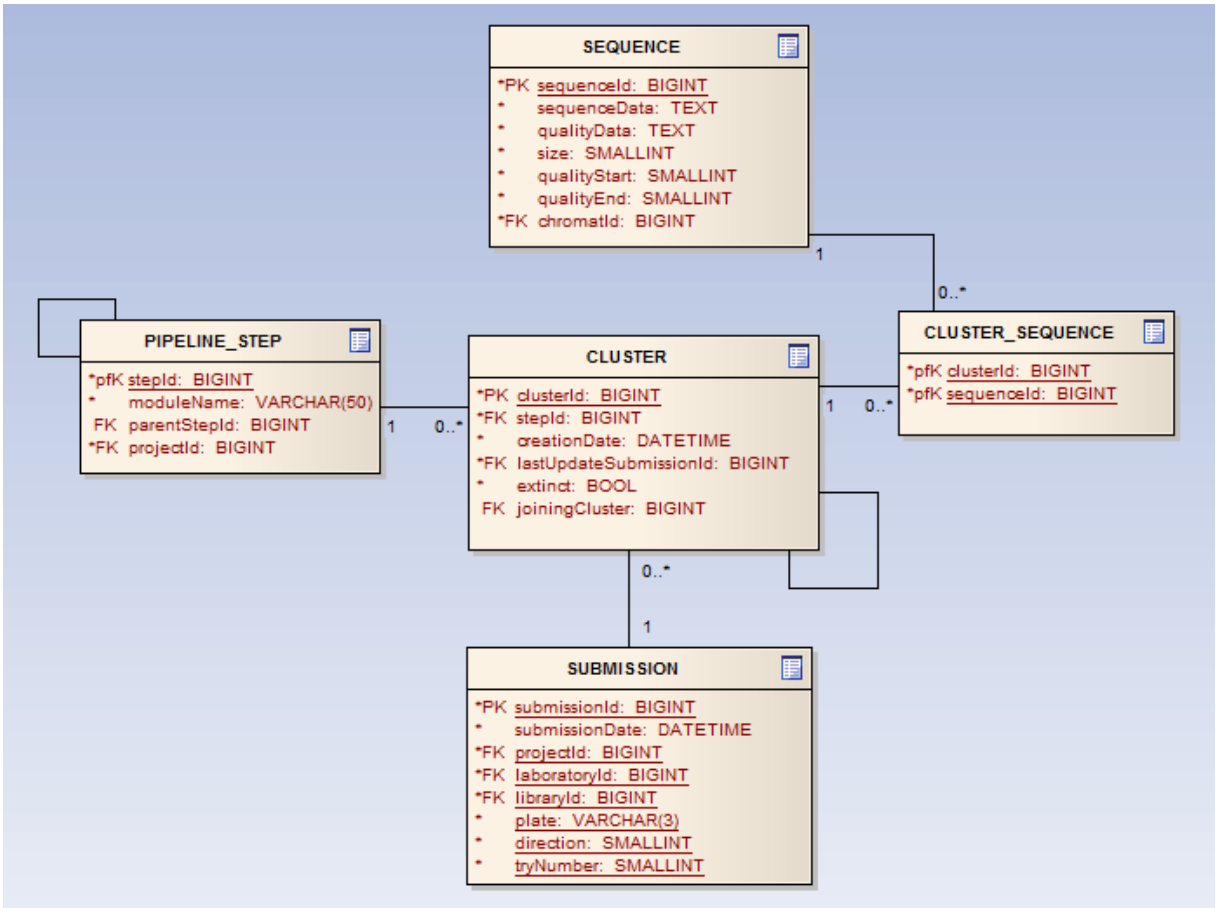

Figura 31 - Modelo lógico de dados para armazenamento de clusters

A tabela *CLUSTER* armazena todos os *clusters* criados em um determinado passo de *pipeline*. Essa tabela também tem campos para informar se o *cluster* está extinto, o *cluster* ao qual ele se uniu caso esteja extinto, e a submissão que efetuou a última atualização. As sequências pertencentes a um cluster são registradas na tabela *CLUSTER_SEQUENCE*. Uma sequência pode pertencer a diferentes *clusters* apenas quando os mesmos foram criados em *pipelines* diferentes.

6.5 CAP3

Os *clusters* formados pelo módulo *Clustering* são agrupamentos de sequências que se alinham. As sequências pertencentes a um *cluster* possuem alto percentual de identidade com as demais sequências do mesmo *cluster*. É possível sobrepor as sequências de modo a formar sequências maiores denominadas *contigs*, em um processo conhecido como montagem ou *assembly*. O **consenso** de um *contig* é formado pelas bases que aparecem com mais frequência em cada posição (ver seção 3.3.3 e Figura 9 para maiores detalhes).

O objetivo da formação de *contigs* é encontrar sequências de genes inteiros a partir da sobreposição de sequências menores. Uma vez que os genes são encontrados, tenta-se descobrir suas funções fazendo-se alinhamentos em bases de genes conhecidos. Essa é a última etapa do *pipeline* construído neste trabalho, explicada na seção 6.6.

O programa externo utilizado para formação de *contigs* é *CAP3*. Este programa aceita como entrada um arquivo FASTA de sequências e outro com valores de qualidade. CAP3 então gera *contigs* sobrepondo as sequências contidas nestes arquivos.

O módulo *CAP3* permite a configuração de dois parâmetros. O parâmetro *overlap_length_cutoff* define o número mínimo de bases sobrepostas para que duas sequências formem um mesmo *contig*. O parâmetro *overlap_identity_cutoff* define o percentual mínimo de identidade entre as sequências formadoras de um *contig*.

A Figura 32 mostra o passo-a-passo do módulo CAP3.

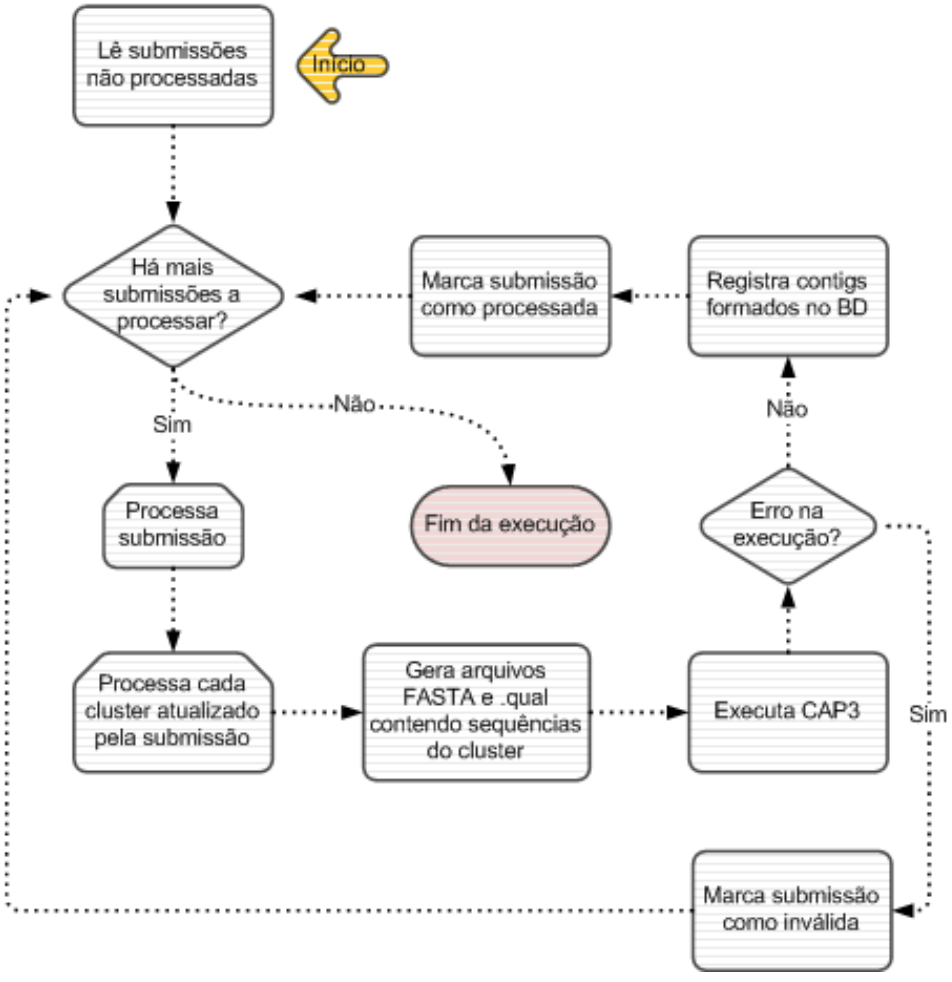

Figura 32 - Passo-a-passo de execução do módulo CAP3

O módulo primeiramente obtém todos os *clusters* atualizados pelo passo anterior e que pertençam a uma submissão não processada pelo passo corrente.

Para cada *cluster* encontrado, o módulo gera um arquivo FASTA de sequências e outro de qualidade em um diretório temporário, contendo todas as sequências presentes no *cluster*. O módulo então executa o programa *CAP3* nesse diretório utilizando a seguinte linha de comando:

```
cap3 cluster.fasta -o <param_overlap_length_cutoff> -p
<param_overlap_identity_cutoff>
```

Nessa linha, "cluster.fasta" é o arquivo FASTA contendo as sequências do *cluster*. <*param_overlap_length_cutoff*> e <*param_overlap_identity_cutoff*> são substituídos pelos parâmetros do módulo com o mesmo nome, e determinam respectivamente os limites de tamanho e percentual de identidade de alinhamento para formação de *contigs*.

O programa *CAP3* gera como resultado um arquivo FASTA contendo os *contigs* formados. Geralmente, apenas um cluster será formado, uma vez que a montagem é feita com sequências de um mesmo *cluster*, e por isso, com alto grau de identidade entre si. Os *contigs* encontrados são armazenados no banco de dados.

Caso não ocorram erros de execução do módulo, a submissão é registrada como processada para a etapa corrente do *pipeline*.

O modelo lógico de dados usado pelo módulo é apresentado na Figura 33. A tabela *CONTIG* armazena os *contigs* individuais, e contém campos que informam o consenso, os valores de qualidade do consenso, o *cluster* gerador do *contig*, o passo de *pipeline* e a submissão nos quais o *contig* foi gerado. A tabela *CONTIG_SEQUENCE* é usada para relacionar as sequências formadoras de um *contig*.

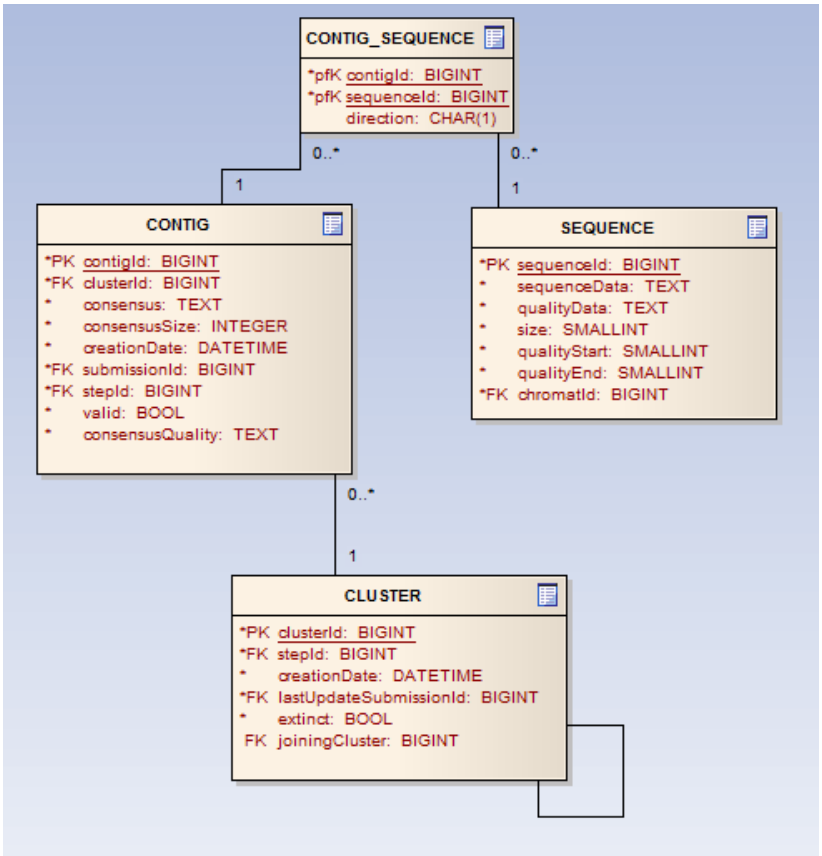

Figura 33 - Modelo lógico de dados para montagem de *contigs*

6.6 Pathways

A última etapa do pipeline efetua atividades de anotação gênica. Seu objetivo é encontrar possíveis funções e vias metabólicas para genes contidos nos *contigs* da etapa anterior. Para isso, são realizadas consultas em bases de dados de proteínas, a fim de se encontrar proteínas que tenham alto grau identidade com as possíveis traduções de um *contig*. Se uma proteína com alto grau de identidade de bases é encontrada, o *contig* possivelmente abriga um gene que produz proteína com funções similares àquela encontrada.

O alinhamento entre *contigs* e proteínas é feito pelo programa externo *BLASTX* [27]. *BLASTX* consulta sequências de nucleotídeos traduzidas em bases de dados de proteínas. Sua função é encontrar proteínas similares àquelas codificadas por sequências de nucleotídeos. *BLASTX* recebe como entrada um arquivo FASTA contendo sequências de nucleotídeos (chamado de *query*) e uma base de dados formatada contendo sequências de proteínas (chamada de *subject*). O resultado é um arquivo em formato tabular contendo em cada linha um alinhamento encontrado, isto é, um par *query-subject*. Outras informações também são apresentadas, como os percentuais de identidade, *e-values*, *score* e outras. Os limites que

determinam quando um alinhamento é válido são configuráveis na linha de comando do programa.

Uma vez encontradas proteínas similares, o próximo passo do módulo é identificar as funções das mesmas e também as vias metabólicas as quais elas pertencem. Neste trabalho, as pesquisas por vias metabólicas são feitas na base de dados Kegg [48]. As vias nesta base são representadas de um modo genérico, aplicável a todos os organismos. Cada via está associada a grupos ortólogos denominados KOs (Kegg Orthology), compostos por proteínas com funções similares entre si.

Para associar proteínas e KOs, este protótipo utiliza uma cópia da base de dados UEKO. UEKO é uma versão enriquecida da base de dados do Kegg, a qual engloba um número maior de associações entre KOs e proteínas (identificadas pelo número de acessão UniProt [26]). UEKO é resultado de um trabalho não publicado desenvolvido por Gabriel Fernandes, pesquisador do Laboratório de Biodados [49] da Universidade Federal de Minas Gerais.

É também mantida uma cópia da tabela de informações de KO, contendo dados importantes de cada KO, tais como nome, descrições e vias metabólicas. Dessa forma, para se identificar as vias metabólicas e funções dos genes encontrados nos *contigs*, é necessário encontrar os grupos ortólogos (KOs) a que eles pertencem. Por meio do KO e do cruzamento de informações nas tabelas citadas acima, é possível deduzir funções e vias metabólicas de grande parte dos genes encontrados.

O módulo *Pathways* permite a configuração do parâmetro *expectation_value*, o qual define o valor *e-value* a ser usado na linha de comando do programa *BLASTX*. O *e-value* representa o número esperado de vezes que um *hit* pode ocorrer ao acaso em uma consulta em toda a base de dados. Portanto, quanto menor o *e-value*, maior a chance que o *hit* não tenha sido ao acaso, e por isso, maior a similaridade entre a sequência *query* e o hit. Estarão na saída de *BLASTX* apenas *hits* cujo *e-value* seja menor que o valor passado no parâmetro *expectation_value*.

A Figura 34 mostra o passo-a-passo de execução do módulo *Pathways*.

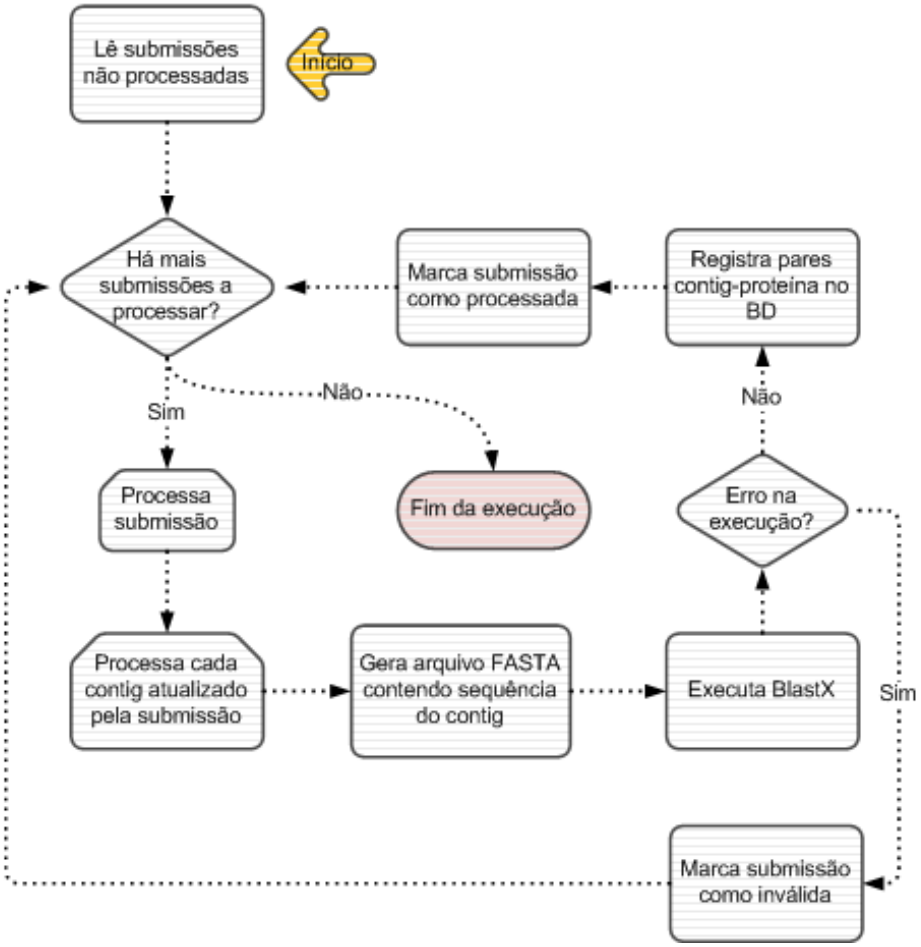

Figura 34 - Passo-a-passo de execução do módulo *Pathways*

O primeiro passo é obter todos os *contigs* gerados pela etapa anterior do pipeline que ainda não foram processados pela etapa corrente.

Em seguida, um arquivo FASTA contendo todos os contigs é gerado em um diretório temporário. *BLASTX* é então é executado com a seguinte linha de comando:

```
blastall -p blastx -i contigs.fasta -d <base_proteinas> -e
<param_expectation_value> -m 8 -b 1
```

Na linha acima, "-p blastx" define que o programa *BLASTX* será usado dentre os programas *BLAST*. "contigs.fasta" é o arquivo FASTA contendo os *contigs* que serve como *query*. <*base_proteinas*> é a base de dados *subject*, um arquivo FASTA contendo as sequências de aminoácidos de todas as proteínas presentes na base de dados UEKO. Esse arquivo fica em um diretório externo configurável conhecido pelo módulo. <*param_expecation_value*> é substituído pelo parâmetro de módulo *expectation_value* e

define o valor máximo de *e-value* que um *hit* poderá ter. "-m 8" determina saída tabular. O parâmetro "-b 1" define que apenas o melhor *hit* aparecerá nos resultados.

O arquivo gerado pela execução de *BLASTX* é lido e os pares *contig*-proteína encontrados são armazenados em uma tabela do banco de dados juntamente com informações do alinhamento, tais como identidade, *e-value* e *score*. O cruzamento dos dados desta tabela com a base de dados UEKO e informações de KO permite descobrir as possíveis funções e vias metabólicas dos genes.

A Figura 35 mostra o modelo lógico de dados utilizado para o módulo *Pathways*.

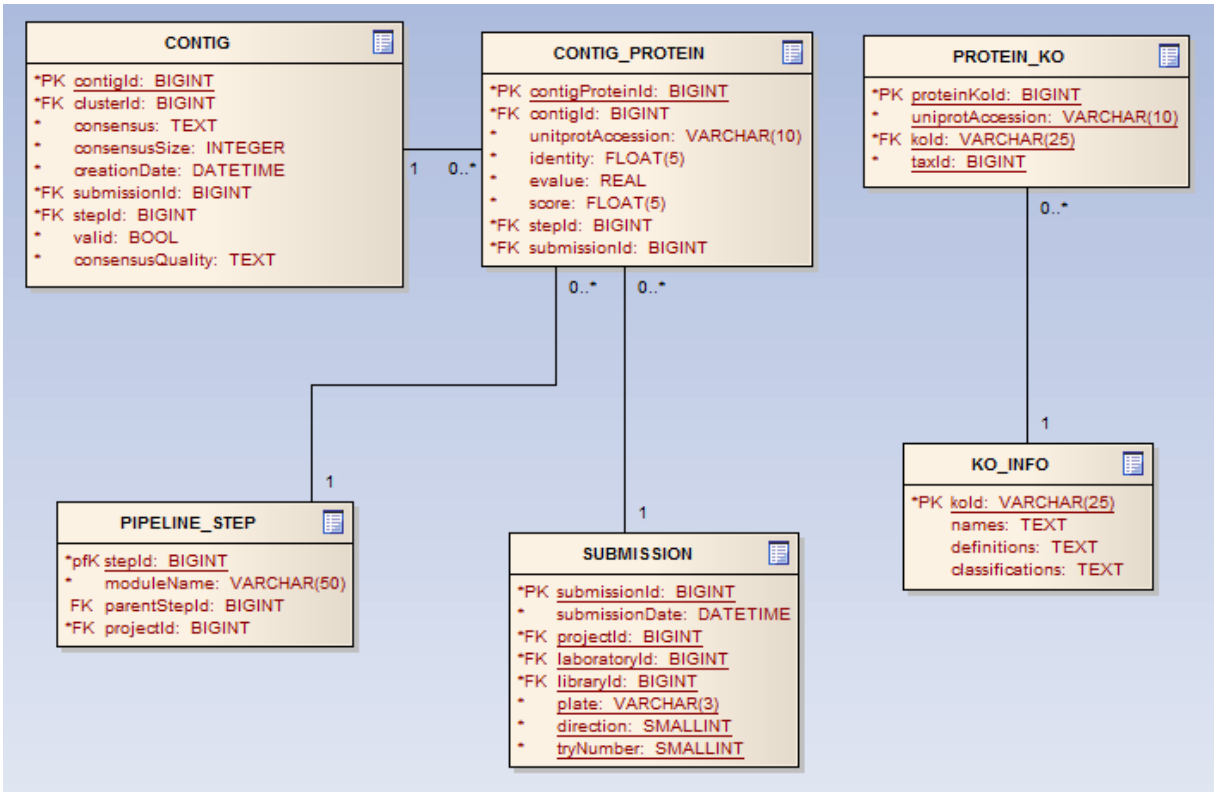

Figura 35 - Modelo lógico de dados utilizado para aferir funções e vias metabólicas

A tabela *PROTEIN_KO* é uma cópia da base de dados UEKO, e contém os relacionamentos entre números de acessão Uniprot e KOs. *KO_INFO* armazena nomes, definições e classificações de cada KO. A tabela *CONTIG_PROTEIN* guarda os alinhamentos realizados pelo programa *BLASTX*, associando *contigs* a números de acessão Uniprot. É possível descobrir os KOs associados aos números de acessão na tabela *PROTEIN_KO*, e então identificar nomes, definições e vias metabólicas na tabela *KO_INFO*.

6.7 Considerações Finais

Neste capítulo, os cinco módulos implementados no protótipo foram apresentados. Os módulos executam diferentes etapas de um pipeline genômico, iniciando em *base-calling* e terminando com a descoberta de funções e vias metabólicas de genes encontrados nas sequências.

É importante lembrar que outros módulos poderiam ainda complementar o pipeline, realizando outras tarefas de anotação gênica, tais como análises filogenéticas dos genes encontrados. O sistema construído permite a fácil integração de novos módulos, por meio da implementação da interface *PipeModule*.

Enfim, a comunicação entre os diferentes módulos cria um sistema robusto e eficiente para análise de dados genômicos.

7. DETALHES DE IMPLEMENTAÇÃO

7.1 Considerações Iniciais

Neste capítulo são apresentados detalhes técnicos da implementação do protótipo. Pretende-se, desse modo, demonstrar como o trabalho pode ser replicado. Na seção 7.2 é apresentada uma visão geral da implementação dos componentes. Nas seções subsequentes, as implementações de cada componente são descritas com maiores detalhes.

7.2 Visão Geral

O protótipo é dividido em três componentes. No componente *Núcleo* se localiza toda a lógica de negócio do *pipeline* e acesso a dados. Esse componente, construído em uma arquitetura orientada a objetos na linguagem Java e sob a plataforma *Spring* [50], utiliza *Hibernate* [51] para persistência.

O componente *Web Service* serve como um *front end* para o *Núcleo.* Apresenta uma interface acessível na linguagem universal XML, contendo um conjunto de operações para inserção de novos dados no sistema, e também consultas. O *Web Service* é construído na linguagem Java e disponibilizado por meio de *engine* Apache Axis2 [52].

O componente *Web* é uma aplicação *Web* construída na plataforma *Ae*, do projeto FAPESP Tidia [53]. Esse componente apresenta interfaces gráficas para interação com o usuário. Apresenta um cliente de *Web service* interno, também construído por meio da tecnologia Axis2, através do qual faz solicitações aos componentes lógicos do sistema. É construído na linguagem Java, fazendo também uso das tecnologias JSF [54], *Spring* e *Hibernate.*

7.3 Componente *Núcleo*

O componente *Núcleo* é totalmente desenvolvido na linguagem Java 5.0 SE. Utiliza o *ORM Hibernate* [51] 3.1.3 para mapeamento objeto-relacional e *Spring 2.0* [50] para controle de injeções de dependência.

O processamento desse componente é a base de lógica e dados do sistema, e é iniciado de duas maneiras:

- Pelo componente *Web service,* para consultas e submissão de novos dados.

- Como um processo em executado em *background* periodicamente pela ferramenta *Cron* [47], quando os *pipelines* são executados.

7.3.1 Camadas

O componente é dividido em duas camadas. A camada de lógica contém todas as regras de negócios da aplicação. Para A camada de persistência é composta pelos *Models*, que representam as entidades, e os *DAOs* que são objetos utilizados para acesso ao banco de dados. A camada de lógica acessa a persistência para armazenamento e acesso a dados. Nas próximas seções são fornecidos mais detalhes do funcionamento de dada componente e camada.

7.3.2 Banco de dados

O SGDB utilizado no componente *Nucleo* foi *MySQL* 5.0 [55]. A escolha foi feita com base na robustez e escalabilidade desse sistema, além de apresentar funcionalidades importantes como suporte a restrições de relacionamentos entre tabelas, o que garante a integridade dos dados.

Os diagramas lógicos utilizados em cada uma das funcionalidades são apresentados ao longo dos capítulos 5 e 6.

7.3.3 Camada Model

A camada *Model* é composta por classes do tipo *POJO (Plain Old Java Objects)* e representa as entidades do domínio na aplicação. Existe uma correlação direta entre as tabelas do banco de dados e essas entidades.

O *ORM Hibernate* 3.1.3 [51] é utilizado na realização do mapeamento objeto-relacional. Através de *Hibernate*, todo o relacionamento entre as tabelas do banco de dados e objetos do sistema é realizado em arquivos XML denominados *mappings* (mapeamentos). Essa é uma funcionalidade útil, pois evita que dados de conexão, nomes de tabelas e campos fiquem inseridos diretamente no código, o que obrigaria uma recompilação em caso de alterações. *Hibernate*, que permite o mapeamento objeto-relacional de forma declarativa (XML), o que diminui a complexidade do código, aumentar o reuso e desacopla o sistema de SGBDs específicos [51].

As entidades da camada *Model* são utilizadas em todas as outras camadas do sistema. Exemplos de *models* são *Chromat, Sequence, PipelineStep, Pipeline*, entre outros.

7.3.4 Camada Dao

A camada *Dao* contém classes que fazem acesso direto ao banco de dados do sistema.

DAO (Data Access Object, ver seção 2.3*)* é um padrão de projetos que separa a interface de acesso a dados de implementações específicas. No protótipo, cada DAO é uma interface com métodos de modificação de dados, como *save, load, update, delete.* A implementação é feita através de interfaces disponibilizadas pelo *ORM Hibernate* e injetadas no sistema pelo *framework Spring*.

A injeção de dependências é um recurso particularmente útil, pois repassa as responsabilidades de instanciação e criação de dependências entre objetos para uma aplicação externa. A configuração das dependências é realizada em arquivos XML.

Através da injeção de dependência propiciada pelo *framework Spring*, é possível integrar outras implementações. Seria possível, por exemplo, criar uma implementação de DAO utilizando JDBC. Este modelo deixa o sistema totalmente independente de implementações específicas de acesso a dados.

Em geral, para cada entidade de *model* existe um DAO específico. Por exemplo, a entidade *Sequence* possui um *SequenceDao*, que tem operações para modificação dos dados de sequências. É possível persistir novas sequências e atualizar aquelas existentes através desse DAO.

A camada *Dao* não é acessada diretamente pelos componentes de negócio da aplicação. Existe uma camada intermediária denominada *Logic,* a qual propicia esse acesso. Esta é uma forma de proteger os dados contra acessos diretos.

7.3.5 Camada Logic

Logic é uma camada intermediária entre os componentes do domínio da aplicação e o modelo de dados. Esta camada contém regras de negócio e operações para acesso a dados.

São criados objetos *Logic* para áreas mais amplas da aplicação. *AssemblyLogic* por exemplo, é responsável pela lógica de negócios de funcionalidades relacionadas com montagem, e acessa diversos DAOs.

Objetos da camada *Logic* também são injetados no sistema como *beans* do *framework Spring*. Essa é uma funcionalidade que permite que apenas uma instância do *bean* seja criada

no ciclo de vida da aplicação. O *framework Spring* faz o controle da instanciação e injeção dos *beans* de lógica em outros *beans* que deles necessitem. O módulo *PhredModule*, por exemplo, necessita do *bean ChromatLogic* para obter cromatogramas do banco de dados. O módulo não precisa instanciar um objeto *ChromatLogic* nesse caso, pois a instância única do mesmo é injetada em *PhredModule* pelo *framework Spring*.

7.3.6 TreeLogic

TreeLogic é um *logic* especial com a funcionalidade de ler configurações de *pipeline* e a partir delas, construir árvores em memória contendo os passos dos *pipelines* lidos.

TreeLogic é chamado quando novas configurações de *pipeline* são inseridas no banco (através do *Web service).* Para cada uma delas, faz o *parsing* do arquivo XML que contém as configurações.

TreeLogic verifica se os módulos descritos na configuração realmente existem. Caso contrário, o *pipeline* não é inserido na árvore.

A árvore de *pipelines* é montada utilizando-se a API de *javax.swing.TreeNode* [56]. Cada objeto desta classe representa um nodo de uma árvore recursiva. Os nodos podem armazenar um objeto do tipo *Object*. Neste caso, é utilizado o objeto *PipelineStep*, que é um filho direto de *Object*. Portanto, cada *TreeNode* é configurado com um *PipelineStep*, que representa um passo de *pipeline*.

A classe *PipelineStep* possui dois atributos: *PipeModule* e um conjunto de *PipelineParam*. Assim que um módulo é encontrado no arquivo de configuração e não se encontra um correspondente na árvore de *pipelines* existente, um objeto do tipo *PipelineStep* é criado utilizando-se o módulo e os parâmetros encontrados na configuração.

Primeiramente, *TreeLogic* carrega toda a árvore de *pipelines* existente em memória. Então, a partir da raiz da árvore, compara gradativamente os passos presentes no arquivo de configuração com passos presentes na árvore. Os passos são reutilizados até que se encontre um nodo na árvore que não possua nenhum passo igual ao passo encontrado no arquivo de configuração. Neste momento, um novo *TreeNode* é criado e inserido como filho do nodo atual. Este *TreeNode* é configurado para carregar o *PipelineStep* criado. A travessia da árvore então prossegue linearmente a partir deste ponto. Ao fim da travessia, o *pipeline* lido do arquivo de configuração estará totalmente incluído na forma de passos na árvore. O último passo deste *pipeline* é armazenado na tabela PIPELINE. Só é necessário saber o último passo para traçar todo o caminho de volta até a raiz, restituindo-se assim o *pipeline* em memória.

Ao fim da leitura dos arquivos de configuração, as configurações já transformadas em *pipelines* armazenadas no banco, e associadas à *pipelines* presentes na tabela PIPELINE.

7.3.7 PipelineExecuter

PipeExecuter é outro *bean* especial, chamado apenas quando o componente *Nucleo* é executado como um processo em *background.* É responsável por fazer a travessia completa da árvore de pipelines, de modo a passar em todos os nodos e executar os passos de *pipeline* neles contidos.

Cada nodo da árvore de *pipelines* implementa a interface *Runnable* da API Java [56]. Dessa forma, os nodos podem ser executados em *threads*. É usada nessa implementação um *pool de threads* para controlar o número de *threads* em execução simultânea. O *pool de threads* é criado a partir do objeto *ThreadPoolExecutor* da API Java 5. Esse *pool de threads* é integrado ao sistema na forma de um *bean*, através do framework *Spring*. Isso garante que o número de *threads* em execução simultânea possa ser configurado em um XML externo. Dessa forma, o sistema pode ser otimizado de acordo com a quantidade de dados tratados. Um volume muito grande de dados pode gerar um número muito grande de *threads* e causar com isso uma queda no desempenho do sistema. Por isso é importante configurar da melhor maneira possível o parâmetro que define o número de *threads* no *pool.*

PipelineExecuter primeiramente lê todos os *PipelineSteps* de forma a carregar a árvore em memória. Em seguida, coloca o nodo raiz na fila de execução do *pool de threads.* A *thread* principal do sistema fica em estado de *sleep* até que o *pool* não tenha mais tarefas a executar. Quando o nodo raiz é executado pelo *pool*, inicia-se um processo recursivo, onde um nodo executa o seu módulo e sem seguida dispara uma *thread* para cada um de seus nodos filhos. Este processo termina com execução de todos os passos da árvore.

Como será mostrado a seguir, cada módulo é configurado como um *bean* no arquivo de configuração de *beans* do *Spring*. Todos esses módulos são implementações da interface *PipeModule,* que apresenta um único método *execute.*

Dessa forma, em cada passo, o nodo busca o módulo associado a ele no *ApplicationContext* do *Spring*. O *framework Spring* então injeta a implementação do módulo no objeto que necessita dele. É importante notar que para o nodo da árvore, é transparente qual é de fato a implementação de módulo que está sendo utilizada: o método *execute* é chamado e então todo o processamento é transferido para o módulo injetado.

7.3.8 Módulos

Conforme descrito no capítulo 6, um módulo representa a unidade de execução de um passo de *pipeline*. Cada uma das diferentes etapas de um *pipeline* genômico deve ter um módulo próprio para execução.

A comunicação entre os módulos é toda feita via banco de dados. Dessa forma, um módulo deve obter os dados gerados pelo módulo anterior no banco, processá-los, e escrever os resultados novamente no banco para que o próximo módulo tenha acesso.

Todos os módulos devem implementar a interface *PipeModule,* que contém o método único *executeModule.* A implementação desse método deve conter toda a lógica de processamento do módulo.

Para que outros componentes tenham acesso aos módulos implementados, é necessário que esses estejam configurados como *beans* no arquivo de configuração do *framework Spring*.

Cada módulo deve ter um identificador (ex: *phred, cap3, seqclean)* que é usado para se obter a implementação correta no *ApplicationContext* do Spring.

7.4 Componente *Web*

O componente *Web* é a parte do protótipo que faz a interação direta com o usuário. É responsável pela criação, configuração e gerenciamento de projetos. Também controla o registro de usuários, assim como seus papéis e permissões dentro do sistema. Através deste componente, o pesquisador é capaz de criar *pipelines* personalizados, enviar cromatogramas e visualizar relatórios com os resultados de seus *pipelines*.

7.4.1 Visão Geral

Esse componente foi construído como uma customização da plataforma Ae. A plataforma Ae, desenvolvida no projeto Tidia FAPESP [53], é um sistema que propicia suporte ao gerenciamento de cursos e projetos de natureza colaborativa. A plataforma provê um conjunto de ferramentas, tais como Chat, Wiki e Forum, que facilitam a comunicação e troca de informações entre integrantes. Um *worksite* é um ambiente restrito a um grupo de usuários, composto de uma seleção de ferramentas feita de acordo com os interesses do grupo. *Worksites* são utilizados por grupos de pesquisa colaborativa, representando o "*site*" do projeto, onde ocorrem a trocas de informação ente os membros. *Worksites* contam ainda com

um sistema de papéis atribuídos a membros que determinam conjuntos de permissões que um usuário possui para acessar funcionalidades específicas.

Por ser uma versão customizada da plataforma *Ae*, o módulo se beneficia com todo o suporte que esta plataforma oferece a projetos de pesquisa colaborativa. Para cada projeto de sequenciamento genômico é criado um *worksite* acessível apenas aos integrantes do projeto. O sistema customiza a plataforma *Ae* de forma a criar um tipo de *worksite* específico para projetos de sequenciamento. Este tipo, denominado "*genome project worksite*", é pré-configurado com quatro possíveis papéis: Líder de Projeto, Coordenador de Grupo de Pesquisa, Pesquisador e Visitante (é importante salientar que um papel no Ae nada mais é do que um conjunto de permissões atribuídas a um usuário). Líder de Projeto é o papel atribuído ao usuário que criou o *worksite* do projeto, e ele tem permissões para atribuir papéis aos demais membros, de acordo com suas tarefas. As permissões e responsabilidades de cada papel serão descritas mais adiante neste texto.

Para que funcione de maneira adequada, um *worksite* do tipo "*genome project worksite*" deve possuir obrigatoriamente três ferramentas essenciais para o desenvolvimento de projetos de sequenciamento genômico: *Project Setup, Chromat Submission e My Pipelines.* Essas ferramentas são partes do componente *Web* incluídas na versão customizada da plataforma *Ae*. Elas serão explicadas em maiores detalhes mais adiante no texto.

7.4.2 Papéis

Na plataforma *Ae*, papéis representam conjuntos de permissões atribuídas a membros de um *worksite*. Tais permissões propiciam acesso a diferentes funcionalidades.

Os papéis criados para um *worksite* do componente *Web* estão dentro de uma hierarquia, sendo que um papel de nível mais alto dentro desta hierarquia engloba todas as permissões do nível inferior e adiciona algumas outras. Os papéis criados pelo componente *Web*, de um nível mais alto para um mais baixo dentro da hierarquia são:

- Líder de Projeto: Papel com o maior número de permissões. É atribuído ao criador do *site*, mas pode ser delegado a outros membros. Tem permissão para criar *worksites* de projeto.

- Coordenador de Grupo de Pesquisa: É capaz de criar um novo grupo de pesquisa e associar pesquisadores a este grupo. É também capaz de adicionar bibliotecas genômicas ao projeto. Possui, além disso, todas as permissões de um Pesquisador.

- Pesquisador: Para que um membro possa participar de um grupo de pesquisa, é necessário que possua no mínimo o papel de pesquisador, o qual tem permissão para criar novos pipelines e submeter cromatogramas. Porém, estas funcionalidades só estão disponíveis quando o pesquisador está associado a um grupo de pesquisa. Consegue visualizar os resultados de *pipelines* criados por integrantes de seu grupo de pesquisa.

- Visitante: Papel com menor número de permissões. É o papel padrão atribuído a um novo membro do *worksite*. Não possui permissões específicas, é capaz apenas de visualizar informações do projeto e resultados abertos publicamente.

7.4.3 Arquitetura

As três ferramentas que integram o componente *Web* são implementadas dentro de uma mesma arquitetura, baseada no uso de três frameworks: JSF [54], *Spring* [50] e *Hibernate* [51].

JSF é uma implementação do padrão de arquitetura MVC (ver seção 2.4). Esse modelo divide a aplicação em três camadas: Modelo, Apresentação e Controle. A camada de Modelo é composta pelos objetos do domínio da aplicação (por exemplo, os objetos de domínio Projeto, Biblioteca e Laboratório) e também pela lógica que modifica e persiste estes objetos. A camada de Apresentação é composta pelas interfaces de interação com o usuário (JSP e HTML). A camada de Controle é composta pelos objetos que processam ações do usuário e organizam o fluxo da aplicação (*Backing Beans*).

A comunicação entre as camadas é feita por meio de objetos gerenciados pelo *framework Spring* [50], denominados *beans. Beans* são objetos criados em instância única, e injetados pelo *framework* em componentes que dependam deles. Esse padrão permite um baixo acoplamento entre os diferentes módulos do sistema, já que as dependências são configuradas arquivos XML. No componente *Web* foram criados três tipos de *beans: backing beans*, *logic beans* e *dao beans.*

Backing beans são *beans* associados diretamente a alguma interface de interação com o usuário (geralmente uma página *Web*). São os elementos constituintes da camada de controle. Recebem dados submetidos pelo usuário e fazem chamadas correspondentes a métodos da camada de modelo. Estas chamadas são feitas através de *logic beans,* os quais são injetados no *backing bean* via *Spring*.

Logic beans são objetos gerenciados, pertencentes à camada de modelo, os quais possuem métodos para consultar ou modificar dados do domínio da aplicação. Tais métodos podem representar operações de persistência ou regras de negócio. Para efetuar operações de persistência, utilizam *dao beans,* que são injetados via *Spring*. Nesse componente, as chamadas e consultas à lógica do sistema são realizadas através do *Web Service.* Por isso, o componente possui um cliente interno que acessa o componente *Web service* para gerenciamento de dados de projeto, submissão de cromatogramas e geração de relatórios. Para isso, utiliza Axis2 [52] .

Dao beans são implementações do padrão de projeto DAO (*Data Access Object,* ver seção 2.2). Possuem métodos que efetuam consultas ou modificações no banco de dados da aplicação. No componente *Web,* os DAOs são implementados fazendo uso do *ORM (Object-relational mapping) Hibernate*, que permite o mapeamento objeto-relacional de forma declarativa (XML).

7.5 Componente *Web Service*

O componente *Web service* é construído na linguagem Java e faz uso do *engine* Axis2 1.4.1 para disponibilização de serviços *Web* provendo acesso à lógica do componente *Núcleo.* O *Web service* dispõe de operações para gerenciamento de projetos, submissão de cromatogramas, consultas e geração de relatórios.

Nos testes realizados neste trabalho, o componente *Web service* foi executado no servidor *Tomcat 5.5* [57].

7.6 Considerações Finais

A implementação do componente *Web* mostrou fornecer uma interface amigável para que integrantes de projetos de sequenciamento possam personalizar seus *pipelines*. A plataforma Ae, que conta com tecnologias *Web* 2.0, é um ambiente bastante compatível com projetos desta natureza, pois fornece ferramentas que auxiliam muito a comunicação entre membros do projeto, além de criar um ambiente compartilhado.

No atual estágio de implementação do protótipo, o modelo apresentado vêm sendo construído com sucesso, e os *pipelines* vem mostrando desempenho satisfatório.

8. ESTUDO DE CASO

8.1 Considerações Iniciais

Para a avaliação da eficiência e dos resultados do *pipeline* desenvolvido neste trabalho foi realizado um estudo de caso utilizando cromatogramas gerados no Laboratório de Biologia Molecular da Universidade Federal de São Carlos.

O laboratório está realizando o sequenciamento de uma biblioteca de cDNA do besouro curculionídeo *Sphenophorus Levis* [58], popularmente conhecido como *bicudo-da-cana*. Essa praga da cana-de-açúcar é responsável por perdas de produção na ordem de 20 a 23 toneladas ao ano por hectare [59]. O estudo das sequências gênicas dessa espécie é muito importante, uma vez que pode levar ao desenvolvimento de novas formas de combate a essa praga no sentido de se minimizar as perdas na produção.

O sequenciamento de *S. levis*, conduzido pelo biólogo Fernando Fonseca P. de Paula e orientado pelo Prof. Dr. Flávio Henrique Silva, produziu até o momento cerca de mil cromatogramas. Os clones de cDNA foram sequenciados individualmente e os cromatogramas gerados foram submetidos ao protótipo para análise.

Neste capítulo, são descritas as etapas realizadas para a configuração do projeto no protótipo, a criação de um *pipeline* para análise e os resultados obtidos após a execução do mesmo.

8.2 Configuração do Projeto

Na ferramenta Configurações de Projeto do componente *Web* do protótipo foi criada uma instância do projeto de sequenciamento de *S. Levis*. Os dados do projeto, tais como nome, descrição, líder e URL, foram preenchidos e submetidos juntamente com a chave de acesso "*slevis*". Também foram cadastrados o laboratório e a biblioteca, à qual foi dado o nome de "SL1". Decidiu-se por usar *UniVec* como banco de vetores.

8.3 Criação do *Pipeline*

O *pipeline* utilizado no estudo de caso foi construído na ferramenta *Meus Pipelines*. Foi configurado com os seguintes parâmetros:

- Módulo *Phred*

  trim_cutoff = 0.032

- Módulo *SeqClean*

  min_length = 100

  min_perc_identity = 96%

- Módulo *Clustering*

  identity_cutoff = 96%

  min_length_coverage = 70%

- Módulo *CAP3*

  overlap_identity_cutoff = 90%

  overlap_length_cutoff = 40

- Módulo *Pathways*

  expectation_value = 1e-10

  filter_query_sequence = true

Ao *pipeline* foi dado o nome de *slevis_pipe_001*.

8.4 Submissão de cromatogramas

Os cromatogramas foram submetidos pela ferramenta *Submissão de Cromatogramas* do componente *Web*. Foi gerado um arquivo *zip* para cada uma das onze placas da biblioteca. Os arquivos *zip* continham 96 cromatogramas, cada um correspondente a um poço da placa (alguns dos arquivos continham menos que 96 cromatogramas, uma vez que os poços que apresentam erros no sequenciamento não são submetidos).

Os arquivos *zip* foram submetidos um a um após o preenchimento do formulário de submissão, no qual também foram informados os números da placa e outros dados.

8.5 Execução do *Pipeline*

O *pipeline* foi iniciado via linha de comando por meio da execução do componente *Núcleo*.

A execução foi realizada em um servidor com processador *Intel Xeon E5430 Quad-core* 2.66 GHz, o qual possui 8 núcleos de processamento. O servidor possuía 3 Gb de memória RAM. O processamento levou cerca de 22 minutos.

8.6 Resultados

Com o fim da execução do *pipeline*, as tabelas do banco de dados ficam repletas de informações geradas na execução de cada etapa. Os relatórios contendo os resultados são construídos a partir de cruzamentos de dados entre essas tabelas.

Nesta seção, são apresentados três relatórios que ilustram de forma significativa os resultados da execução. O relatório geral mostra uma visão geral dos resultados, com as principais estatísticas. O relatório de agrupamento mostra o número de sequências dos 30 maiores *clusters* formados na execução do *pipeline*. O último relatório é certamente o de maior interesse dos pesquisadores, mostrando homologias encontradas entre os *contigs* gerados e proteínas conhecidas, separadas por vias metabólicas.

8.6.1 Relatório Geral

```
Número de submissões processadas: 11
Número de cromatogramas: 1026
Número de sequências geradas por Phred: 1026
Tamanho médio das sequências: 678,33 bases
Número de sequência invalidadas por SeqClean: 310
Sequências válidas após SeqClean: 716
Número de clusters formados: 307
Número de contigs formados: 316
Tamanho médio dos contigs: 519,8 bases
Número de contigs com hits da base UEKO: 171
```

8.6.2 Relatório de Agrupamento

Este relatório mostra os quinze maiores agrupamentos de sequências encontrados, isto é, grupos de sequências que apresentam sobreposição (*clusters)*. A Tabela 1 lista os resultados do relatório.

Cluster	Número de Sequências
18	90
46	64
14	49
24	23
29	13
65	11
20	10
57	10
21	9
40	9
42	8
3	7
11	7
35	7
52	6

Tabela 1- Quinze maiores agrupamentos (*clusters)*

8.6.3 Relatório de Homologias e Vias Metabólicas

O relatório de vias metabólicas lista as vias associadas aos possíveis genes presentes nos *contigs.* Ele foi construído da maneira descrita a seguir.

Na última etapa do *pipeline*, os 171 *contigs* encontrados foram alinhados por BLASTX na base de proteínas UEKO. Foram encontradas 160 proteínas cujas sequências de bases se alinham com um ou mais consensos, indicando prováveis homologias. Foram obtidos então os grupos ortólogos (KOs) associados às proteínas, os quais totalizaram 165. Os grupos ortólogos, por sua vez, faziam parte de um total de 136 vias metabólicas distintas.

O relatório foi gerado de modo a agrupar os *contigs* encontrados por via metabólica. No Apêndice A, uma parte do relatório é apresentada, onde são listadas as maiores vias metabólicas encontradas. Para cada via, são também listados os *contigs* associados e as funções dos possíveis genes presentes.

8.7 Considerações Finais

Os resultados do estudo de caso mostram que o protótipo realizou com sucesso as diferentes etapas do *pipeline*. Ao final da análise das onze placas submetidas, 171 *contigs* apresentaram uma ou mais homologias com genes presentes na base UEKO, relacionando-os a possíveis funções e vias metabólicas conhecidas. Esse número representa aproximadamente 50% do total de *contigs* gerados na análise.

Para efeito de comparação, os *contigs* gerados foram alinhados pelo programa *BLASTX* na base de proteínas do NCBI conhecida como *nr*. Essa base inclui todas as sequências das principais bases de peptídeos disponíveis (*GenBank CDS translations, RefSeq Proteins, PDB, SwissProt, PIR e PRF*), sendo frequentemente atualizada. O resultado da execução de *BLASTX* mostrou que 224 *contigs* se alinharam com genes depositados em *nr,* representando aproximadamente 66% do total de *contigs.*

É importante lembrar que UEKO é uma base cujo objetivo é o reconhecimento das vias metabólicas associadas à sequências de peptídeos. Desse modo, não estão presentes nessa base proteínas cujas vias metabólicas ainda não estão cadastradas no banco de dados do Kegg. Neste estudo, 50% dos *contigs* foram associados a uma ou mais vias metabólicas. Tendo em vista que o percentual de *contigs* que obtiveram *hits* em *nr (*que contém todas as proteínas catalogadas) foi de 66%, esse pode ser considerado um resultado muito bom. Isso porque aproximadamente 74% dos *contigs* que tiveram *hit* em *nr* foram associados a vias metabólicas.

A identificação de vias metabólicas é de grande importância para o reconhecimento de genes de interesse para a comunidade científica. O gene que codifica a *catepsina L,* por exemplo, está presente nas amostras submetidas ao *pipeline,* tendo sido inclusive já depositado no *GenBank* (com *accesion number* B8Y319). A análise efetuada no protótipo identificou a presença desse e outros genes, os quais participam da via metabólica nomeada *Peptidases* (ver Apêndice A).

O sistema apresentado nesse trabalho é extensível, permitindo a fácil inclusão de novos módulos. Em trabalhos futuros o autor incluirá módulos para comparações em outras bases de dados de proteínas além de UEKO, tais como KOG [60] e GOA [61], o que certamente aumentará o número de homologias encontradas na análise.

9. CONCLUSÕES

Neste trabalho, foi realizado um estudo dos principais pipelines de análise genômica e transcriptômica disponíveis atualmente na literatura. A partir desse estudo, foi desenvolvido um protótipo de *pipeline* com base em *Web Services* com o objetivo de suprir as deficiências apresentadas nos sistemas estudados. Um estudo de caso foi realizado para validação do protótipo construído.

Revisando a literatura, verificou-se que a maioria dos *pipelines* existentes carece de funcionalidades importantes para o desenvolvimento de projetos de sequenciamento. Geralmente fazem análises pontuais de um conjunto fixo de cromatogramas. No entanto, projetos de sequenciamento normalmente têm natureza progressiva e colaborativa. É importante que os sistemas de análise sejam capazes de acompanhar e gerar resultados parciais das análises ao longo do desenvolvimento do projeto. Também é útil a existência de um ambiente colaborativo onde os diferentes laboratórios envolvidos possam contribuir com novos dados e cromatogramas.

Outro ponto de deficiência em grande parte dos pipelines estudados é a impossibilidade da configuração dos parâmetros da análise. O suporte a múltiplos *pipelines* com diferentes configurações permite a otimização da análise com base no organismo em estudo e nos objetivos do projeto.

A inclusão de novos programas e módulos ao *pipeline* é outra funcionalidade ausente em diversos *pipelines* estudados. Novos programas estão surgindo constantemente, o que justifica a importância desse recurso.

O protótipo desenvolvido neste trabalho faz o acompanhamento do progresso de projetos de sequenciamento ao longo de todo o seu desenvolvimento. Para isso, recebe cromatogramas, realiza análises dos dados parciais e emite relatórios com os resultados. Também armazena informações do projeto, laboratórios envolvidos e bibliotecas.

A comunicação com o servidor de processamento é realizada via *Web service*, oferecendo uma interface na linguagem universal XML que permite que aplicações cliente em plataformas heterogêneas submetam dados e realizem operações e consultas.

O protótipo dá suporte a múltiplos *pipelines* em um mesmo projeto. Os *pipelines* são configurados através de arquivos XML em formato específico, no qual o pesquisador define os programas a parâmetros a utilizar. A execução dos *pipelines* é realizada em paralelo por

meio de um *pool de threads,* o que aumenta a eficiência dividindo a carga de processamento em servidores com mais de um núcleo.

Uma aplicação cliente foi desenvolvida na plataforma colaborativa *Ae,* permitindo que diferentes grupos de pesquisa envolvidos no sequenciamento criem pipelines, visualizem resultados e troquem informações sobre o andamento do projeto. Tal característica vai de encontro à natureza colaborativa de projetos de sequenciamento.

Outra característica do protótipo desenvolvido é a extensibilidade. Cada etapa do pipeline é encapsulada em um módulo. Novos módulos podem ser facilmente inseridos sem a necessidade de recompilação de todo o sistema, bastando para isso que o mesmo implemente uma interface específica. A inserção no sistema é realizada declarativamente em arquivos XML.

Por fim, o estudo de caso realizado neste trabalho mostrou que o pipeline desenvolvido é eficiente e apresenta bons resultados. A integração do sistema com a base UEKO possibilita a descoberta das principais vias metabólicas das quais os genes presentes nas amostras fazem parte.

Desse modo, o protótipo construído apresenta funcionalidades que suprem as deficiências identificadas nos sistemas correlatos, atingindo os objetivos do projeto. O sistema mostrou que poderia ser usado em projetos de sequenciamento envolvendo grande número de laboratórios.

9.1 Trabalhos decorrentes

Uma decorrência natural deste trabalho é a implementação de novos módulos. Foram desenvolvidos cinco módulos com as principais etapas de análise gênica. No entanto, módulos que realizem análises filogenéticas e outras etapas de anotação também são desejáveis. Outra possibilidade é a integração de outras bases de proteínas, além de UEKO, para a busca de homologias.

Também é planejada a integração do sistema como um serviço no CENABID (Centro Nacional Centro Nacional de Processamento em Bioinformática de Intenso Desempenho). O CENABID se propõe a centralizar e disponibilizar serviços de Bioinformática pela Web, fazendo uso do alto desempenho computacional dos CENAPAD (Centro Nacional de Processamento de Alto Desempenho). Os CENAPAD compõem o programa SINAPAD, uma rede de centros de computação de alto desempenho, geograficamente distribuídos, instituída pelo Ministério da Ciência e Tecnologia (MCT). O protótipo desenvolvido pode ser

facilmente integrado a esse centro, uma vez que é operado por meio de *Web services* e construído de forma modular.

10. REFERÊNCIAS

1. CURBERA, F. et al. Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. **IEEE Internet computing**, v. 6, p. 86-93, 2002.

2. W3C. **SOAP Specifications**, 2007. Disponivel em: <http://www.w3.org/TR/soap/>. Acesso em: 7 Agosto 2009.

3. W3C. **Web Services Description Language (WSDL)**, 2001. Disponivel em: <http://www.w3.org/TR/wsdl>. Acesso em: 7 Agosto 2009.

4. OASIS. UDDI Specification. **xml.org**, 2006. Disponivel em: <http://uddi.xml.org/specification>. Acesso em: 7 Agosto 2009.

5. VOORMANN, H. Statemaster Encyclopedia. **Statemaster Encyclopedia - Web service**, 2003. Disponivel em: <http://www.statemaster.com/encyclopedia/Web-service>. Acesso em: 7 Agosto 2009.

6. ALUR, D.; MALKS, D.; CRUPI, J. **Core J2EE Patterns:** Best Practices and Design Strategies (2nd Edition) (Sun Core Series). 2. ed. [S.l.]: Prentice Hall PTR, 2003.

7. SINGH, I. et al. **Designing Enterprise Applications with the J2EE(TM) Platform (2nd Edition) (Java Series)**. 2. ed. [S.l.]: Prentice Hall PTR, 2002.

8. GOETZ, B. Java theory and practice: Thread pools and work queues. **IBM Website**, 2002. Disponivel em: <http://www.ibm.com/developerworks/java/library/j-jtp0730.html>. Acesso em: 15 Fevereiro 2009.

9. KNUTH, D. E. **Art of Computer Programming, Volume 1:** Fundamental Algorithms (3rd Edition) (Art of Computer Programming Volume 1). 3. ed. [S.l.]: Addison-Wesley Professional, 1997.

10. HOHPE, G.; WOOLF, B. **Enterprise Integration Patterns:** Designing, Building, and Deploying Messaging Solutions (Addison-Wesley Signature Series). [S.l.]: Addison-Wesley Professional, 2003.

11. LARADE, K.; STOREY, K. B. Constructing and screening a cDNA library. Methods for identification and characterization of novel genes expressed under conditions of environmental stress. **Methods Mol Biol**, v. 410, p. 55-80, 2008.

12. SANGER, F.; NICKLEN, S.; COULSON, A. R. DNA sequencing with chain-terminating inhibitors. 1977. **Biotechnology**, v. 24, p. 104-108, 1992.

13. NAGARAJ, S. H.; GASSER, R. B.; RANGANATHAN, S. A hitchhiker's guide to expressed sequence tag (EST) analysis. **Brief Bioinform**, v. 8, p. 6-21, 2007.

14. NCBI. Fasta format description. **Site do NCBI**. Disponivel em: <http://www.ncbi.nlm.nih.gov/blast/fasta.shtml>. Acesso em: 14 Fevereiro 2009.

15. EWING, B. et al. Base-calling of automated sequencer traces using phred. I. Accuracy assessment. **Genome Res**, v. 8, p. 175-185, 1998.

16. SAHA, S. et al. Computational Approaches and Tools Used in Identification of Dispersed Repetitive DNA Sequences. **Tropical Plant Biology**, v. 1, p. 85-96, 2008.

17. LABORATORY OF PHIL GREEN. **Phred, Phrap and Consed**. Disponivel em: <http://www.phrap.org/phredphrapconsed.html#block_phrap>. Acesso em: 13 Fevereiro 2009.

18. DFCI. **TGI Software Tools**. Disponivel em: <http://compbio.dfci.harvard.edu/tgi/software/>. Acesso em: 13 Fevereiro 2009.

19. SMITH, T.; WATERMAN, M. Identification of common molecular subsequences. **J. Mol. Bwl**, v. 147, p. 195-197, 1981.

20. ZHANG, Z. et al. A greedy algorithm for aligning DNA sequences. **J Comput Biol**, v. 7, p. 203-214, 2000.

21. HUANG, X.; MADAN, A. CAP3: A DNA sequence assembly program. **Genome Res**, v. 9, p. 868-877, 1999.

22. CHEN, T.; SKIENA, S. S. A case study in genome-level fragment assembly. **Bioinformatics**, v. 16, p. 494-500, 2000.

23. PERTEA, G. et al. TIGR Gene Indices clustering tools (TGICL): a software system for fast clustering of large EST datasets. **Bioinformatics**, v. 19, p. 651-652, 2003.

24. WU, C.; NEBERT, D. W. Update on genome completion and annotations: Protein Information Resource. **Human Genomics**, v. 1, p. 229-233, 2004.

25. BOECKMANN, B. et al. The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. **Nucleic acids research**, v. 31, p. 365, 2003.

26. APWEILER, R. et al. UniProt: the Universal Protein knowledgebase. **Nucleic Acids Res**, v. 32, p. 115-119, 2004.

27. ALTSCHUL, S. F. et al. Basic local alignment search tool. **J Mol Biol**, v. 215, p. 403-410, 1990.

28. MAO, C. et al. ESTAP--an automated system for the analysis of EST data. **Bioinformatics**, v. 19, p. 1720-1722, 2003.

29. D'AVILA, A. M. et al. GARSA: genomic analysis resources for sequence annotation. **Bioinformatics**, v. 21, p. 4302-4303, 2005.

30. PAQUOLA, A. C. et al. ESTWeb: bioinformatics services for EST sequencing projects. **Bioinformatics**, v. 19, p. 1587-1588, 2003.

31. HOON, S. et al. Biopipe: a flexible framework for protocol-based bioinformatics analysis. **Genome Res**, v. 13, p. 1904-1915, 2003.

32. SHAH, S. P. et al. Pegasys: software for executing and integrating analyses of biological sequences. **BMC Bioinformatics**, v. 5, p. 40-40, 2004.

33. POTTER, S. C. et al. The Ensembl analysis pipeline. **Genome Res**, v. 14, p. 934-941, 2004.

34. HUBBARD, T. et al. The Ensembl genome database project. **Nucleic Acids Res**, v. 30, p. 38-41, 2002.

35. STAJICH, J. E. An Introduction to BioPerl. **Methods Mol Biol**, v. 406, p. 535-548, 2007.

36. DURHAM, A. M. et al. EGene: a configurable pipeline generation system for automated sequence analysis. **Bioinformatics**, v. 21, p. 2812-2813, 2005.

37. NAGARAJ, S. H. et al. ESTExplorer: an expressed sequence tag (EST) assembly and annotation platform. **Nucleic Acids Res**, v. 35, p. 143-147, 2007.

38. ZOPE.ORG. **Zope.org**, 2009. Disponivel em: <http://www.zope.org>. Acesso em: 4 Agosto 2009.

39. FORMENT, J. et al. EST2uni: an open, parallel tool for automated EST analysis and database creation, with a data mining web interface and microarray expression data integration. **BMC Bioinformatics**, v. 9, p. 5-5, 2008.

40. OTTO, T. D. et al. ChromaPipe: a pipeline for analysis, quality control and management for a DNA sequencing facility. **Genet Mol Res**, v. 7, p. 861-871, 2008.

41. LIANG, C. et al. MAGIC-SPP: a database-driven DNA sequence processing package with associated management tools. **BMC Bioinformatics**, v. 7, p. 115-115, 2006.

42. KUMAR, C. G. et al. ESTIMA, a tool for EST management in a multi-project environment. **BMC Bioinformatics**, v. 5, p. 176-176, 2004.

43. PARKINSON, J. et al. PartiGene--constructing partial genomes. **Bioinformatics**, v. 20, p. 1398-1404, 2004.

44. AYOUBI, P. et al. PipeOnline 2.0: automated EST processing and functional data sorting. **Nucleic Acids Res**, v. 30, p. 4761-4769, 2002.

45. O'REILLY, T. What Is Web 2.0. **O'Reilly Media**, 2005. Disponivel em: <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>. Acesso em: 15 Fevereiro 2009.

46. NCBI. UniVec Database. **Site do NCBI**. Disponivel em: <http://www.ncbi.nlm.nih.gov/VecScreen/UniVec.html>. Acesso em: 23 Julho 2009.

47. GNU. GNU Cron. **The GNU Operating System**. Disponivel em: <http://www.gnu.org/software/gcron/main.html>. Acesso em: 28 Julho 2009.

48. OGATA, H. et al. KEGG: Kyoto Encyclopedia of Genes and Genomes. **Nucleic Acids Res**, v. 27, p. 29-34, 1999.

49. LAB. BIODADOS UFMG. **Lab. Biodados UFMG**. Disponivel em: <http://www.biodados.icb.ufmg.br>. Acesso em: 28 Julho 2009.

50. SPRINGSOURCE.ORG. **Spring Framework Website**, 2009. Disponivel em: <http://www.springsource.org/>. Acesso em: 15 Fevereiro 2009.

51. HIBERNATE.ORG. **Hibernate Website**, 2009. Disponivel em: <http://www.hibernate.org/>. Acesso em: 15 fev. 2009.

52. THE APACHE SOFTWARE FOUNDATION. Apache Axis2. **The Apache Software Foundation Website**, 2008. Disponivel em: <http://ws.apache.org/axis2/>. Acesso em: 8 Agosto 2009.

53. FAPESP. **Portal do Tidia-Ae**, 2008. Disponivel em: <http://www.tidia-ae.iv.fapesp.br>. Acesso em: 15 Fevereiro 2009.

54. SUN MICROSYSTEMS. **JavaServer Faces Technology**, 2008. Disponivel em: <http://java.sun.com/javaee/javaserverfaces/>. Acesso em: 15 Fevereiro 2009.

55. SUN MICROSYSTEMS. **MySQL Website**, 2009. Disponivel em: <http://www.mysql.com/>. Acesso em: 15 Fevereiro 2009.

56. SUN MICROSYSTEMS. **Java 2 Platform SE 5.0**, 2004. Disponivel em: <http://java.sun.com/j2se/1.5.0/docs/api/>. Acesso em: 15 Fevereiro 2009.

57. THE APACHE SOFTWARE FOUNDATION. Apache Tomcat. **The Apache Software Foundation Website**, 2009. Disponivel em: <http://tomcat.apache.org/>. Acesso em: 11 Agosto 2009.

58. FONSECA, F. P. P. **Construção de um banco de ESTs e sua utilização para clonagem e estudos funcionais de proteases intestinais de Sphenophorus levis, o bicudo da**

cana-de-açúcar. UFSCAR. [S.l.]. Em andamento.

59. CENTRO CANAGRO JOSÉ CORAL. Cartilha de Pragas - Bicudo. **Site do Centro Canagro de Piracicaba e Região "José Coral"**, 2005. Disponivel em: <http://www.cana.com.br/biblioteca/cartilha_praga/bicudo.pdf>. Acesso em: 16 Agosto 2009.

60. TATUSOV, R. L. et al. The COG database: an updated version includes eukaryotes. **BMC bioinformatics**, v. 4, p. 41, 2003.

61. BARRELL, D. et al. The GOA database in 2009--an integrated Gene Ontology Annotation resource. **Nucleic Acids Research**, 2008.

62. WATSON N, J. D.; CRICK, F. H. Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid. **Nature**, v. 171, p. 737-738, 1953.

63. SALZBERG, S. L. et al. Interpolated Markov models for eukaryotic gene finding. **Genomics**, v. 59, p. 24-31, 1999.

64. HARTWELL, L. et al. **Genetics:** From Genes to Genomes. 3. ed. [S.l.]: McGraw-Hill Science/Engineering/Math, 2006.

65. GAMMA, E. et al. **Design Patterns:** Elements of Reusable Object-Oriented Software (Addison-Wesley Professional Computing Series). illustrated edition. ed. [S.l.]: Addison-Wesley Professional, 1994.

66. BENSON, D. A. et al. GenBank. **Nucleic Acids Res**, v. 37, p. 26-31, 2009.

67. ALBERTS, B. et al. **Molecular Biology of the Cell**. 5. ed. [S.l.]: Garland Science, 2007.

68. BAIROCH, A. et al. The Universal Protein Resource (UniProt). **Nucleic Acids Res**, v. 33, p. 154-159, 2005.

69. BORODOVSKY, M.; MCININCH, J. GeneMark: parallel gene recognition for both DNA strands. **Computers & Chemistry**, 17, n. 19, 1993. 123-133.

70. SOARES-COSTA, A. et al. **Purification and Characterization of a Midgut Cysteine Peptidase from the Sugarcane Weevil Sphenophorus levis**. XXXVIII Annual Meeting

of SBBq. Águas de Lindóia: [s.n.]. 2009.

Apêndice A — RELATÓRIO DE VIAS METABÓLICAS

Id do Contig	Uniprot Accession	KO	Nomes	Definições	Ident. (%)	E-value
Protein Families; Metabolism; Peptidases [BR:ko01002]						
21	B4HVI6	K09646	SCPEP1	serine carboxypeptidase 1 [EC:3.4.16.-]	58,5	6E-50
37	Q5TRG5	K08776	NPEPPS	puromycin-sensitive aminopeptidase [EC:3.4.11.-]	76,62	2E-71
55	Q91X79	K01326	E3.4.21.36, ELA1	pancreatic elastase I [EC:3.4.21.36]	40,38	8E-15
101	Q6DJ90	K09640	TMPRSS9	transmembrane protease, serine 9 [EC:3.4.21.-]	39,69	6E-15
128	B4J3I5	K01310	E3.4.21.1	chymotrypsin [EC:3.4.21.1]	42,75	2E-23
133	O46030	K01365	E3.4.22.15, CTSL	cathepsin L [EC:3.4.22.15]	41,57	3E-33
133	O46030	K09600	CTSM	cathepsin M [EC:3.4.22.-]	41,57	3E-33
177	Q6A1I1	K01366	E3.4.22.16, CTSH	cathepsin H [EC:3.4.22.16]	43,75	6E-29
178	O60235	K09641	TMPRSS11D	transmembrane protease, serine 11D [EC:3.4.21.-]	34,9	2E-14
196	Q6P1S3	K01262	E3.4.11.9, pepP	X-Pro aminopeptidase [EC:3.4.11.9]	69,47	1E-33
204	Q5RZV1	K01365	E3.4.22.15, CTSL	cathepsin L [EC:3.4.22.15]	38,12	9E-34
209	B7P3P0	K01363	E3.4.22.1, CTSB	cathepsin B [EC:3.4.22.1]	48,08	2E-43
226	B4KUA2	K01290	E3.4.17.1	carboxypeptidase A [EC:3.4.17.1]	37,28	2E-21
232	Q6ISU5	K01311	E3.4.21.2, CTRC	chymotrypsin C [EC:3.4.21.2]	40,41	9E-20
232	Q6ISU5	K01326	E3.4.21.36, ELA1	pancreatic elastase I [EC:3.4.21.36]	40,41	9E-20
232	Q6ISU5	K01345	E3.4.21.70, ELA3	pancreatic endopeptidase E [EC:3.4.21.70]	40,41	9E-20
232	Q6ISU5	K01346	E3.4.21.71, ELA2	pancreatic elastase II [EC:3.4.21.71]	40,41	9E-20
297	B4LJE9	K01365	E3.4.22.15, CTSL	cathepsin L [EC:3.4.22.15]	42,26	3E-52
298	A9U938	K01365	E3.4.22.15, CTSL	cathepsin L [EC:3.4.22.15]	40,24	3E-11
298	A9U938	K09600	CTSM	cathepsin M [EC:3.4.22.-]	40,24	3E-11
300	B6PUP9	K01365	E3.4.22.15, CTSL	cathepsin L [EC:3.4.22.15]	41,71	1E-31
300	B6PUP9	K09600	CTSM	cathepsin M [EC:3.4.22.-]	41,71	1E-31
301	B6PUQ2	K01365	E3.4.22.15, CTSL	cathepsin L [EC:3.4.22.15]	41,38	4E-35
302	B6PUQ2	K01365	E3.4.22.15, CTSL	cathepsin L [EC:3.4.22.15]	43,62	5E-66
306	P91922	K01290	E3.4.17.1	carboxypeptidase A [EC:3.4.17.1]	34,12	5E-40
308	Q8WSH3	K01365	E3.4.22.15, CTSL	cathepsin L [EC:3.4.22.15]	46,32	2E-26
308	Q8WSH3	K09600	CTSM	cathepsin M [EC:3.4.22.-]	46,32	2E-26
312	Q8WR56	K01379	E3.4.23.5, CTSD	cathepsin D [EC:3.4.23.5]	70,81	5E-62
318	Q3V5Y3	K01363	E3.4.22.1, CTSB	cathepsin B [EC:3.4.22.1]	48,84	8E-44
326	B4KNT4	K09645	CPVL	vitellogenic carboxypeptidase-like protein [EC:3.4.16.-]	60,32	2E-39
333	P35041	K01312	E3.4.21.4, PRSS1, PRSS2, PRSS3	trypsin [EC:3.4.21.4]	36,57	2E-19
333	P35041	K01312	E3.4.21.4, PRSS1, PRSS2, PRSS3	trypsin [EC:3.4.21.4]	36,57	2E-19
342	Q6DJC1	K01368	E3.4.22.27, CTSS	cathepsin S [EC:3.4.22.27]	39,13	4E-27
342	Q6DJC1	K01371	E3.4.22.38, CTSK	cathepsin K [EC:3.4.22.38]	39,13	4E-27

Protein Families; Genetic Information Processing; Ribosome [BR:ko03011]						
22	Q4GXJ8	K02868	RP-L11e, RPL11	large subunit ribosomal protein L11e	98,46	5E-69
33	Q6EUX6	K02950	RP-S12, rpsL	small subunit ribosomal protein S12	95,9	3E-65
33	Q6EUX6	K02973	RP-S23e, RPS23	small subunit ribosomal protein S23e	95,9	3E-65
40	Q56FF2	K02981	RP-S2e, RPS2	small subunit ribosomal protein S2e	93,75	4E-25
40	Q56FF2	K02988	RP-S5, rpsE	small subunit ribosomal protein S5	93,75	4E-25
84	Q56FD3	K02872	RP-L13Ae, RPL13A	large subunit ribosomal protein L13Ae	87,38	8E-49
117	Q4GXE0	K02898	RP-L26e, RPL26	large subunit ribosomal protein L26e	87,04	2E-46
135	Q6EV08	K02941	RP-LP0, RPLP0	large subunit ribosomal protein LP0	92,75	3E-106
140	Q8ITC5	K02908	RP-L30e, RPL30	large subunit ribosomal protein L30e	84,85	6E-44
180	Q4GXH1	K02874	RP-L14, rplN	large subunit ribosomal protein L14	100	6E-68
180	Q4GXH1	K02894	RP-L23e, RPL23	large subunit ribosomal protein L23e	100	6E-68
183	Q4GXL9	K02936	RP-L7Ae, RPL7A	large subunit ribosomal protein L7Ae	82,64	5E-59
186	Q4GXM3	K02937	RP-L7e, RPL7	large subunit ribosomal protein L7e	89,41	2E-84
223	Q4GXU7	K02987	RP-S4e, RPS4	small subunit ribosomal protein S4e	97,44	4E-38
227	Q95NR0	K02927	RP-L40e, RPL40	large subunit ribosomal protein L40e	98,88	6E-43
344	Q95V32	K02991	RP-S6e, RPS6	small subunit ribosomal protein S6e	90,97	2E-71

Protein Families; Genetic Information Processing; Chaperones and folding catalysts [BR:ko03110]						
30	O16303	K09503	DNAJA2	DnaJ homolog, subfamily A, member 2	49,57	5E-42
50	A6YM15	K04079	htpG, HSP90A	molecular chaperone HtpG	93,55	4E-76
74	Q5CCL7	K09553	STIP1	stress-induced-phosphoprotein 1	57,76	4E-33
80	Q28ZF4	K09542	CRYAB	crystallin, alpha B	47,83	4E-15
106	B4NML7	K09542	CRYAB	crystallin, alpha B	49,66	7E-31
107	A0FDR1	K03671	trxA	thioredoxin 1	64,76	8E-35
149	Q9U639	K03283	HSPA1_8	heat shock 70kDa protein 1/8	87,27	2E-21
149	Q9U639	K09490	HSPA5, BIP	heat shock 70kDa protein 5	87,27	2E-21
150	Q5ZM98	K03283	HSPA1_8	heat shock 70kDa protein 1/8	89,17	1E-56
150	Q5ZM98	K04043	dnaK	molecular chaperone DnaK	89,17	1E-56
154	A9JSW0	K09569	FKBP2	FK506-binding protein 2 [EC:5.2.1.8]	53,85	8E-34
207	A6ZID0	K03283	HSPA1_8	heat shock 70kDa protein 1/8	92,96	4E-73
207	A6ZID0	K09490	HSPA5, BIP	heat shock 70kDa protein 5	92,96	4E-73
335	B6PF95	K09580	PDIA1, P4HB	prolyl 4-hydroxylase, beta polypeptide [EC:5.3.4.1]	62,24	1E-42

Protein Families; Cellular Processes and Signaling; Cytoskeleton proteins [BR:ko04812]						
48	A7UIK9	K07374	TUBA	tubulin alpha	99,36	4E-88
72	O93510	K05768	GSN	gelsolin	52	2E-17
113	Q9NA03	K05692	ACTB_G1	actin beta/gamma 1	100	3E-90
113	Q9NA03	K10354	ACTA1	actin, alpha skeletal muscle	100	3E-90
113	Q9NA03	K10355	ACTF	actin, other eukaryote	100	3E-90
156	Q2YDQ3	K07374	TUBA	tubulin alpha	100	1E-51
165	Q8T7V0	K05692	ACTB_G1	actin beta/gamma 1	100	2E-85
165	Q8T7V0	K10354	ACTA1	actin, alpha skeletal muscle	100	2E-85
165	Q8T7V0	K10355	ACTF	actin, other eukaryote	100	2E-85
236	Q8T8B2	K07375	TUBB	tubulin beta	100	7E-72
239	B6NKM7	K10365	CAPZB	capping protein (actin filament) muscle Z-line, beta	85,29	1E-28

Metabolism; Energy Metabolism; Oxidative phosphorylation [PATH:ko00190]						
124	B6D8Y2	K03878	ND1	NADH dehydrogenase I subunit 1 [EC:1.6.5.3]	69,23	7E-60
127	Q14V44	K00412	CYTB, petB	ubiquinol-cytochrome c reductase cytochrome b subunit [EC:1.10.2.2]	76,19	1E-64
129	O21495	K02261	COX2	cytochrome c oxidase subunit II [EC:1.9.3.1]	74,17	1E-56
141	A8QZJ3	K03883	ND5	NADH dehydrogenase I subunit 5 [EC:1.6.5.3]	74,29	4E-55
159	B6D8X0	K03879	ND2	NADH dehydrogenase I subunit 2 [EC:1.6.5.3]	56	3E-50
172	P31400	K02117	ATPVA, ntpA	V-type H+-transporting ATPase subunit A [EC:3.6.3.14]	88,28	1E-62
172	P31400	K02145	ATPeVA, ATP6A1	V-type H+-transporting ATPase subunit A [EC:3.6.3.14]	88,28	1E-62
263	B6D8Y1	K00412	CYTB, petB	ubiquinol-cytochrome c reductase cytochrome b subunit [EC:1.10.2.2]	75,86	6E-72
265	Q14V50	K02262	COX3	cytochrome c oxidase subunit III [EC:1.9.3.1]	65,22	2E-65
265	Q14V50	K02276	coxC	cytochrome c oxidase subunit III [EC:1.9.3.1]	65,22	2E-65
270	B6D8Y1	K00412	CYTB, petB	ubiquinol-cytochrome c reductase cytochrome b subunit [EC:1.10.2.2]	75,86	6E-72
272	Q14V50	K02262	COX3	cytochrome c oxidase subunit III [EC:1.9.3.1]	65,22	2E-65
272	Q14V50	K02276	coxC	cytochrome c oxidase subunit III [EC:1.9.3.1]	65,22	2E-65
277	B6D8Y1	K00412	CYTB, petB	ubiquinol-cytochrome c reductase cytochrome b subunit [EC:1.10.2.2]	75,86	6E-72
279	Q14V50	K02262	COX3	cytochrome c oxidase subunit III [EC:1.9.3.1]	65,22	2E-65
279	Q14V50	K02276	coxC	cytochrome c oxidase subunit III [EC:1.9.3.1]	65,22	2E-65
284	B6D8Y1	K00412	CYTB, petB	ubiquinol-cytochrome c reductase cytochrome b subunit [EC:1.10.2.2]	75,86	6E-72
286	Q14V50	K02262	COX3	cytochrome c oxidase subunit III [EC:1.9.3.1]	65,22	2E-65
286	Q14V50	K02276	coxC	cytochrome c oxidase subunit III [EC:1.9.3.1]	65,22	2E-65
309	B6RZV9	K02126	ATPeF0A, MTATP6	F-type H+-transporting ATPase subunit a [EC:3.6.3.14]	64,75	3E-47
314	Q1HRQ7	K02111	ATPF1A, atpA	F-type H+-transporting ATPase subunit alpha [EC:3.6.3.14]	83,65	1E-68
314	Q1HRQ7	K02132	ATPeF1A, ATP5A1	F-type H+-transporting ATPase subunit alpha [EC:3.6.3.14]	83,65	1E-68
316	B6D8X7	K03883	ND5	NADH dehydrogenase I subunit 5 [EC:1.6.5.3]	62,66	4E-79

Metabolism; Carbohydrate Metabolism; Starch and sucrose metabolism [PATH:ko00500]						
66	Q16WF4	K05350	bglB	beta-glucosidase [EC:3.2.1.21]	64,15	1E-20
99	B4LQJ1	K01187	E3.2.1.20, malZ	alpha-glucosidase [EC:3.2.1.20]	60,1	2E-65
118	Q16WF4	K05350	bglB	beta-glucosidase [EC:3.2.1.21]	49,43	4E-42
126	P47180	K01184	E3.2.1.15	polygalacturonase [EC:3.2.1.15]	48,28	3E-37
160	Q8I9K6	K01176	E3.2.1.1, amyA, malS	alpha-amylase [EC:3.2.1.1]	76,65	9E-75
217	B0WNN6	K05350	bglB	beta-glucosidase [EC:3.2.1.21]	57,63	8E-35
323	Q0CDS9	K01193	E3.2.1.26, sacA	beta-fructofuranosidase [EC:3.2.1.26]	42,98	2E-17
341	B3MVE8	K01187	E3.2.1.20, malZ	alpha-glucosidase [EC:3.2.1.20]	50,83	8E-25

Metabolism; Carbohydrate Metabolism; Glycolysis / Gluconeogenesis [PATH:ko00010]						
25	Q7Q981	K00016	LDH, ldh	L-lactate dehydrogenase [EC:1.1.1.27]	76	1E-72
38	B0WEB5	K00134	GAPDH, gapA	glyceraldehyde 3-phosphate dehydrogenase [EC:1.2.1.12]	85,5	7E-60
38	B0WEB5	K01803	TPI, tpiA	triosephosphate isomerase (TIM) [EC:5.3.1.1]	85,5	7E-60
123	B0W2T1	K00161	PDHA, pdhA	pyruvate dehydrogenase E1 component subunit alpha [EC:1.2.4.1]	70,55	1E-57
143	B4LVK3	K01689	ENO, eno	enolase [EC:4.2.1.11]	75,76	9E-67
171	B4N1E0	K00128	E1.2.1.3	aldehyde dehydrogenase (NAD+) [EC:1.2.1.3]	56,62	2E-43
171	B4N1E0	K00129	E1.2.1.5	aldehyde dehydrogenase (NAD(P)+) [EC:1.2.1.5]	56,62	2E-43
334	Q6PPI0	K01623	ALDO, fbaB	fructose-bisphosphate aldolase, class I [EC:4.1.2.13]	84,71	1E-73