5 Uma proposta de compactação para o BlastP

Como parte do objetivo de propor uma solução que utilize técnicas de compactação para dados de biosseqüências é estudar e definir uma estratégia de compactação que permita reduzir o tempo de latência das aplicações biológicas na leitura de dados de biosseqüências. Devido ao grande volume de dados dos bancos de dados de biosseqüências, usar técnicas de banco de dados, tais como estratégias de método de acesso e gerência de memória são as mais adequadas para dar suporte a uma estratégia de compactação de dados.

É importante ressaltar que usar a tecnologia de banco de dados aqui não se traduz simplesmente em usar um SGBD conhecido e disponível. Caso fosse adotada esta estratégia seria necessária a recodificação de várias das aplicações que lidam com biosseqüências para que acessassem um SGBD ao invés de arquivos texto ou semi-estruturado.

Foi decidido adotar como estratégia a definição de uma forma de persistência compactada para biosseqüências. E para aplicá-la ao programa BlastP foi necessário agregar funções de gerência de memória e métodos de acesso específicos para lidar com este tipo de dado. Desta maneira, a solução funcionará como mecanismo provedor de dados para as aplicações que lidam com biosseqüências.

Esta decisão se deve ao fato de que a gerência de memória e o método de acesso específico permitem adotar estratégias variadas para os dados compactados e provê-los de forma descompactada para as aplicações, alterando minimamente os algoritmos internos das aplicações. Esta escolha foi feita dada à complexidade da aplicação usada para verificar na prática o comportamento da solução, já que BlastP contém várias decisões de implementação baseadas em conhecimento biológico, e a estratégia busca preservar esse conhecimento, ao contrário de outras que penalizam o resultado em prol de melhor desempenho do programa[DBL+00]. Ou seja, altera-se o programa porém o algoritmo é mantido.

Dada a variedade de soluções de compactação existentes, a adoção de uma solução provedora de dados se baseia nos tempos adequados para descompactar e prover dados para a aplicação.

5.1 A solução proposta

Com esta solução proposta pretende-se validar o uso da técnica de compactação de dados em domínios onde as aplicações são intensivas sobre os dados. Em especial, o foco esteve nas aplicações que precisam lidar com grandes volumes de dados com um alto grau de redundância. Aplicações biológicas e dados biológicos se enquadram exatamente dentro deste contexto, ou seja, aplicações intensivas sobre grandes volumes de dados os quais apresentam alto grau de redundância.

Mais especificamente, a proposta é fornecer uma representação compactada para dados de biosseqüências para reduzir o número de operações de E/S do programa BlastP.

A compactação é feita sobre o arquivo de seqüências gerado pelo aplicativo FORMATDB previamente à execução do programa BlastP. Como parte da solução é incluído no programa um gerente de memória para controlar a descompactação em tempo de execução, que, baseado nas propostas de compactação estudadas, e apresentadas no capítulo 4, será testado em diferentes variantes em relação ao momento de realizar realmente a descompactação.

Desta maneira, a proposta apresentada aqui discute duas estratégias distintas para o tratamento dos dados compactados. As abordagens são:

- **ingênua**: descompactar o arquivo de seqüências inteiro antes da primeira utilização do mesmo.
- **leitura em blocos**: compactar em blocos. Esta opção pode ter variantes para a descompressão, já que os dados podem ser descompactados de uma só vez da perspectiva da aplicação que chama a descompactação, mudando somente o algoritmo de compressão. E a outra opção é fazer a chamada da descompressão também em blocos, de acordo com a demanda da aplicação principal.

A Figura 13 ilustra a interação entre os principais componentes da proposta que será detalhada nas próximas seções. Os componentes envolvidos são: aplicação biológica, gerência de memória para dados compactados, fontes

de dados compactados, compactador de dados biológicos e fonte de dados biológicos.

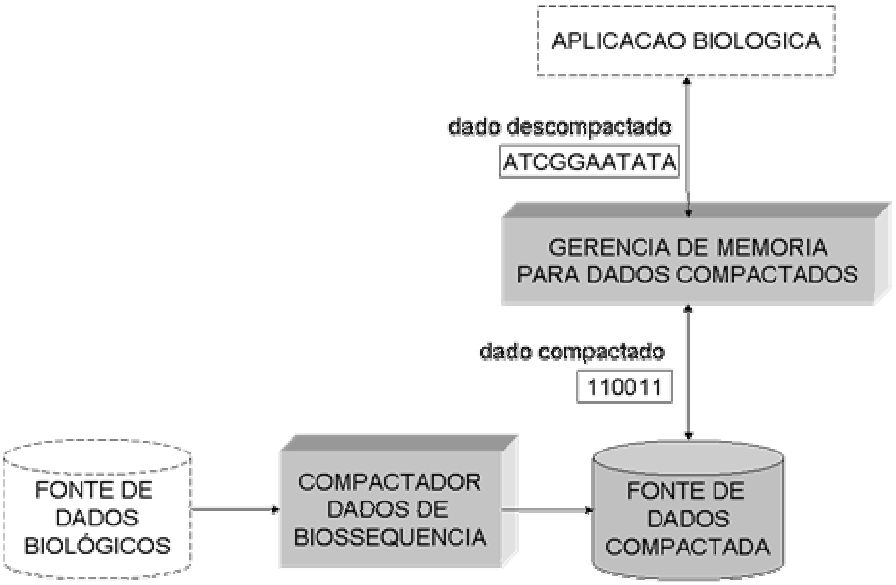

Figura 13 – Principais Componentes da Solução Proposta

Por aplicação biológica entende-se toda e qualquer aplicação que faz uso de dados biológicos, como por exemplo, um programa que lida com dados de biosseqüências. Da mesma forma, as fontes de dados biológicos representam qualquer informação útil para aplicações biológicas.

A solução apresentada neste capítulo propõe a intermediação entre os pedidos de dados de uma determinada aplicação biológica e uma fonte de dados biológica. Ou seja, a gerência de memória recebe as solicitações da aplicação e disponibiliza os dados para aplicação de forma descompactada. Para isso, os dados biológicos precisam ser compactados previamente. No entanto, esta compactação precisará ser executada somente uma única vez, já que após a compactação os dados serão apenas lidos pela aplicação alvo. Além disso, as fontes de dados de biosseqüências públicas sofrem pouca modificação dos dados existentes o que privilegia este tipo de estratégia.

Em função do enorme espectro de aplicações existentes para a área de biologia assim como as fontes de dados, foi decidido utilizar como aplicação biológica deste estudo uma implementação do programa BLAST dada a sua importância na obtenção de conhecimento biológico sobre as seqüências ainda não conhecidas. Além disso, a aplicação BLAST possui a característica de ser

intensiva sobre os dados o que a torna uma excelente candidata para o uso da proposta de gerência de dados compactados.

A decisão pela descompactação gradual exigirá a criação de blocos de dados compactados de tal forma que os blocos possam ser descompactados de maneira isolada. Esse fato tenderá a reduzir a taxa de compressão dos dados e igualmente influenciará no total de descompactações que o gerente de memória deverá executar. É importante ressaltar que o tamanho do bloco a ser compactado é fixo e não o tamanho do bloco após a sua compactação visto que cada bloco poderá ter uma taxa de compressão distinta.

5.2 Detalhamento da Proposta de Solução

Nesta seção é apresentada a solução proposta descrevendo as decisões tomadas durante a sua implementação. No apêndice E pode ser encontrado o código final com comentários.

Como detalhado inicialmente neste capítulo, o estudo sobre a natureza dos dados e a análise das operações mais freqüentes sobre dados de biosseqüências orienta este estudo para a adoção de uma estratégia de gerência de memória aliada a um método de acesso específico para dados de biosseqüências. Além disso, esses métodos de acesso exigem um modelo de persistência que deve lidar com dados compactados. Desta forma, a implementação da solução é dividida em duas partes, doravante chamados de módulos. O primeiro módulo trata da compactação de dados e o segundo da gestão de memória e acesso aos dados compactados.

A Figura 14 apresenta uma visão global dos módulos implementados. Foi utilizada nesta representação a notação UML, cujos símbolos representam pacotes de artefatos de software. A solução proposta é composta por dois pacotes, que foram ressaltados na figura, o pacote Gerente Memória e o pacote Compactador. O pacote do NCBI_BLAST representa o conjunto de artefatos de software que implementam as funcionalidades do programa NCBI-BLAST.

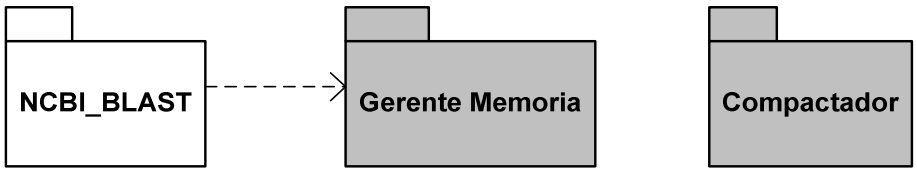

Figura 14 - Módulos Globais

O pacote Gerente Memória contém as funcionalidades de gerência de memória e métodos de acesso a dados compactados. Este pacote é utilizado pelo pacote NCBI-BLAST para ler os dados de biosseqüências compactados e fornecê-los de forma descompactada. Desta maneira o pacote NCBI_BLAST possui uma relação de dependência com o pacote de Gerencia Memória. Por outro lado, o pacote Compactador é usado para compactar os dados de biosseqüências.

Módulo Compactador de dados

O módulo compactador de dados foi implementado usando uma estratégia de compactação reversível baseada em estatísticas, na qual os dados sofrem um pré-processamento, conforme descrito no capítulo 4. Esta estratégia se mostrou adequada para a compactação de arquivos de biosseqüências em tempo aceitável de acordo com [AZM+02]. O pré-processamento é utilizado para ordenar os dados lexicograficamente aumentando a probabilidade de obter a melhor taxa de compressão. Foi obtida uma implementação pronta desta estratégia em [IC06] que combinava os algoritmos BWT e MTF em um único programa e utilizava uma implementação existente do algoritmo de Huffman.

A Figura 15 apresenta um exemplo de como o compactador de dados pode ser usado. É sabido que o programa NCBI-BLAST não processa diretamente os arquivos contendo as informações sobre as seqüências (1). Desta maneira se faz necessário que o arquivo original seja processado previamente por um programa chamado FORMATDB (2) que possui a função de formatá-lo de acordo com a necessidade do NCBI-BLAST. Apesar do FORMATDB gerar três arquivos contendo cada um deles informações importantes para o NCBI-BLAST, nesta proposta é relevante somente lidar com arquivos que contenham as biosseqüências propriamente ditas. Neste caso, o arquivo de interesse é o arquivo cuja extensão é denominada de PSQ. Este

arquivo utiliza um *byte* para representar cada letra de aminoácido (3) e separa as seqüências por um *byte* representando o valor zero.

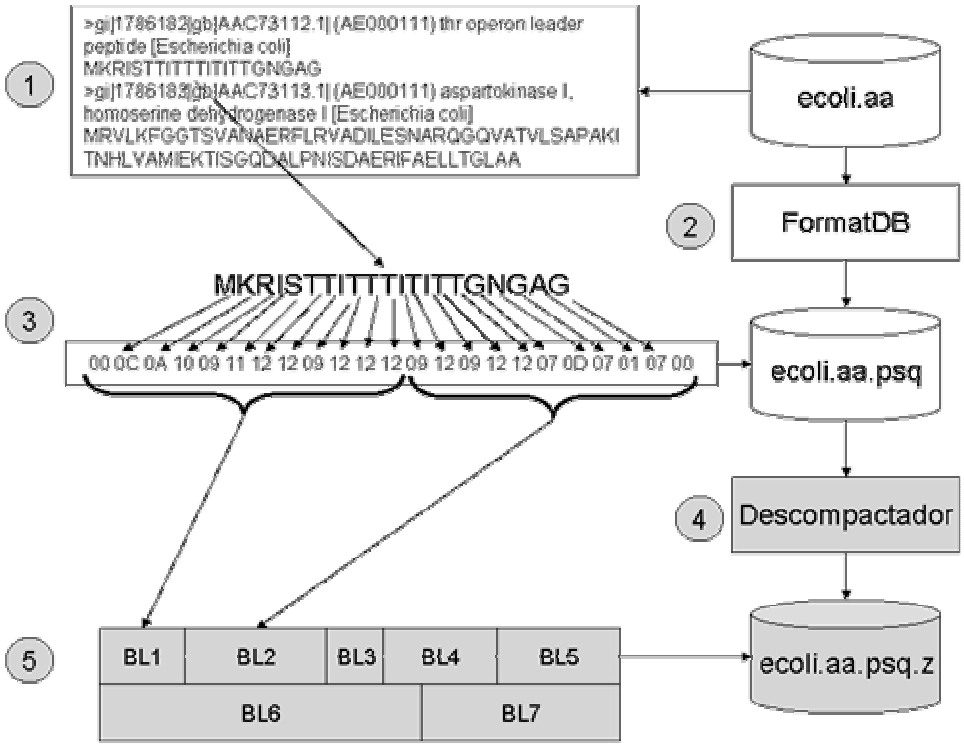

Figura 15 - Compactação dos Dados de Biosseqüências

De acordo com a Figura 15, o módulo compactador utiliza o arquivo PSQ (4) para gerar um novo arquivo com as seqüências compactadas. Este módulo deve ser executado logo após a execução do FORMATDB. A estratégia de compactação adotada lê os dados das biosseqüências gravadas no arquivo PSQ e gera blocos compactados das mesmas (5). O tamanho do bloco deve ser definido no momento da compactação e poderá influenciar na performance da execução do BLAST, pois determinará o número de execuções de descompactação e assim como a taxa de compressão atingida. É importante observar que os tamanhos dos blocos gerados poderão ser irregulares, ou seja, não possuirão o mesmo tamanho, visto que blocos de biosseqüência original podem possuir um grau de redundância. A variabilidade no tamanho dos blocos influenciará no tempo de descompactação dado que este tempo é produto do tamanho do bloco compactado.

O módulo compactador foi implementado através da codificação de um programa escrito na linguagem C e da biblioteca de funções que implementam o

algoritmo de Huffman. A Figura 16 apresenta sob a forma de modelo de classes o programa CompactaSequencia responsável pela compactação das biosseqüências . É representada também a biblioteca de funções do algoritmo de Huffman sob a forma de um pacote de funções. Foi utilizada a relação de dependência para mostrar que o programa CampactaSequencia depende das funções do pacote Huffman para realizar as suas funcionalidades.

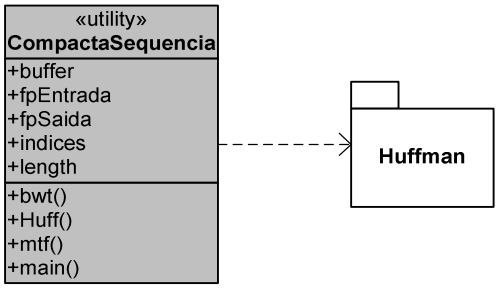

Figura 16 - Módulo Compactador

Na Figura 17 é ilustrada a ordem de execução das funções do programa CompactaSequencia. Um diagrama de seqüência da UML é utilizado para representar esta dinâmica. Desta maneira, após a solicitação de um usuário para a compactação de certo arquivo, serão chamadas as funções bwt, mtf e Huff. Essas funções serão chamadas a cada bloco de biosseqüências lido até que o arquivo de entrada tenha sido integralmente lido.

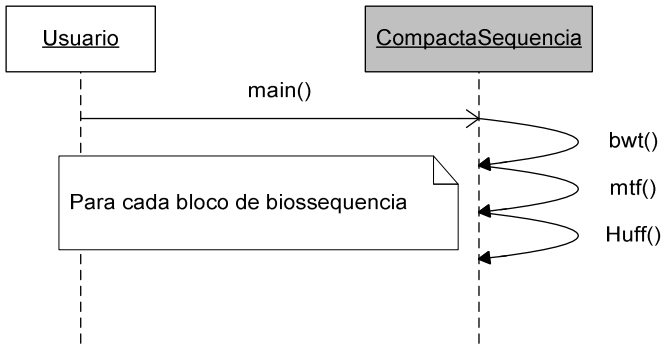

Figura 17 - Seqüência das Chamadas das Funções de Compactador

Apesar de ter sido utilizado um programa de compactação de dados existente, foi necessário modificar grande parte deste programa para adequá-lo às necessidades do módulo de gerência de memória. Para isso foram substituídas as leituras em disco por operações de acesso em memória principal. Além disso, mediante o acompanhamento do programa pelo aplicativo Windows Task Manager e pelo total de execução obtido, foi constatado que a implementação do algoritmo de Huffman não se mostrou adequada para o uso em memória principal, já que realizava um uso intenso da memória disponível e um número grande de acessos a disco. Desta maneira, a implementação foi substituída por uma outra implementação mais adequada [H05].

Módulo Gerência de Memória

O módulo gerência de memória é composto por dois programas: *GerenteMemoria* e *Huffman*, conforme apresentado na Figura 18. Estes programas são representados pelos símbolos de classes UML, preenchidas com a cor cinza. O programa *GerenteMemoria* é o programa principal do módulo e é utilizado por apenas dois programas pertencentes ao pacote que implementa o NCBI-BLAST: *ncbiMem* e *Blast*. O *ncbiMem* implementa todas as funcionalidades associadas às operações de memória utilizadas pelo programa NCBI-BLAST, como detalhado no capítulo 3.

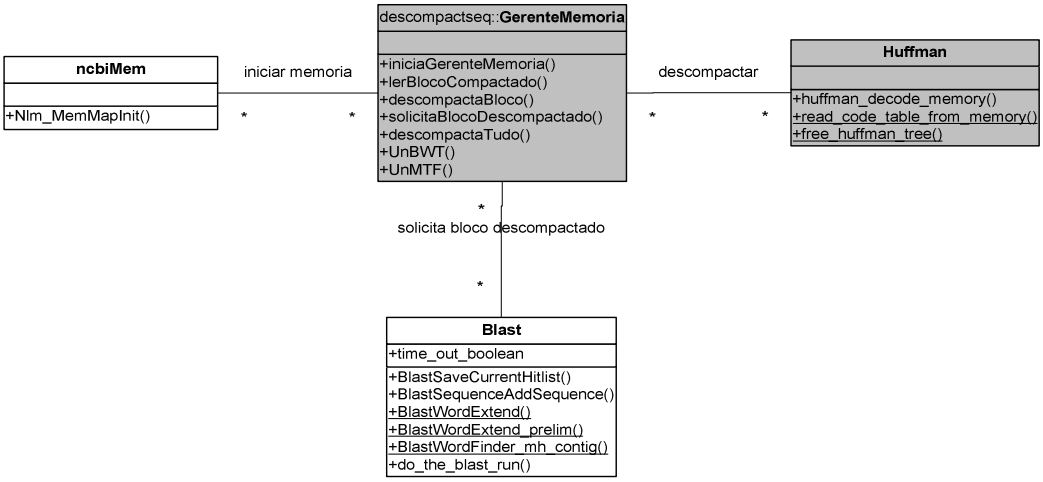

Figura 18 - Gerência de Memória

Durante a integração do módulo Gerência de Memória e o programa NCBI-BLAST, foi necessário inserir uma chamada a *iniciaGerenteMemoria* do programa *GerenteMemoria* dentro da função *MemMapInit* do programa *ncbiMem*. Esta alteração foi necessária para permitir que o módulo Gerente de memória pudesse tomar ciência das estruturas de memória utilizadas pelo NCBI-BLAST. Outra mudança necessária foi a alteração dos comandos que usavam ponteiros para área de memória onde eram colocadas as seqüências descompactadas. Neste caso, foi necessária a identificação e substituição das operações de ponteiros por chamadas da função *solicitaBlocoDescompactado* do programa *GerenteMemoria*. Como essas operações estavam agrupadas em um módulo do BLAST foi possível detectá-las facilmente.

A dinâmica das chamadas de função entre os programas *ncbiMem*, *Blast*, *GerenteMemoria* e *Huffman* é ilustrada na Figura 19. Nesta figura é possível observar que o *GerenteMemoria* é iniciado através de uma chamada ao programa *ncbiMem*. A função *iniciaGerenteMemoria* é a responsável pela iniciação do buffer de memória onde serão fornecidas as biosseqüências descompactadas. Além disso, esta função realiza a leitura do primeiro bloco disponibilizando-o imediatamente em memória para uso pelo programa BLAST posteriormente.

A descompactação de um bloco compactado é realizada através da chamada à função *lerBlocoDescompactado*. Esta função carrega para memória o próximo bloco compactado a ser processado para uma área de memória específica de blocos compactados. Em seguida, inicia o processo de descompactação chamando a função *descompactaBloco*. Esta função por sua vez chama a função *huffman_decode_memory* que descompacta o bloco e coloca o resultado em uma outra área de memória, a qual será passada para a próxima função a ser executada. A função *UnMTF* lê os dados descompactados pela função que implementa o algoritmo de Huffman e coloca-os na ordem original antes da compactação. É importante lembrar que os dados sofreram uma reordenação para aumentar a probabilidade de aumento na taxa de compactação. A função *UnBWT* é a última função a ser chamada, recebendo o resultado do algoritmo *unMTF* e utilizando a tabela de transformação T mais um índice o qual indica qual a linha desta tabela que contém o primeiro caractere do texto original. Com esses dados, o algoritmo BWT consegue refazer o texto original.

Durante a execução do programa BLAST são executadas inúmeras chamadas ao método *solicitaBlocoDescompactado* do programa Gerente de memória (vide Figura 19). Esta função recebe como parâmetro um ponteiro para o *buffer* de memória e o utiliza para determinar quantos blocos deverão ser descompactados para suprir a necessidade da aplicação. Sendo assim, esta função fornece sob demanda os dados descompactados para o programa BLAST. Foi usada também uma estratégia de leitura antecipada para evitar um retardo na espera do BLAST por dados ainda não descompactados. Quando o ponteiro para o *buffer* atinge 100 *bytes*, valor definido aleatoriamente, do final do *buffer* descompactado, a função realiza a descompactação do próximo bloco imediatamente.

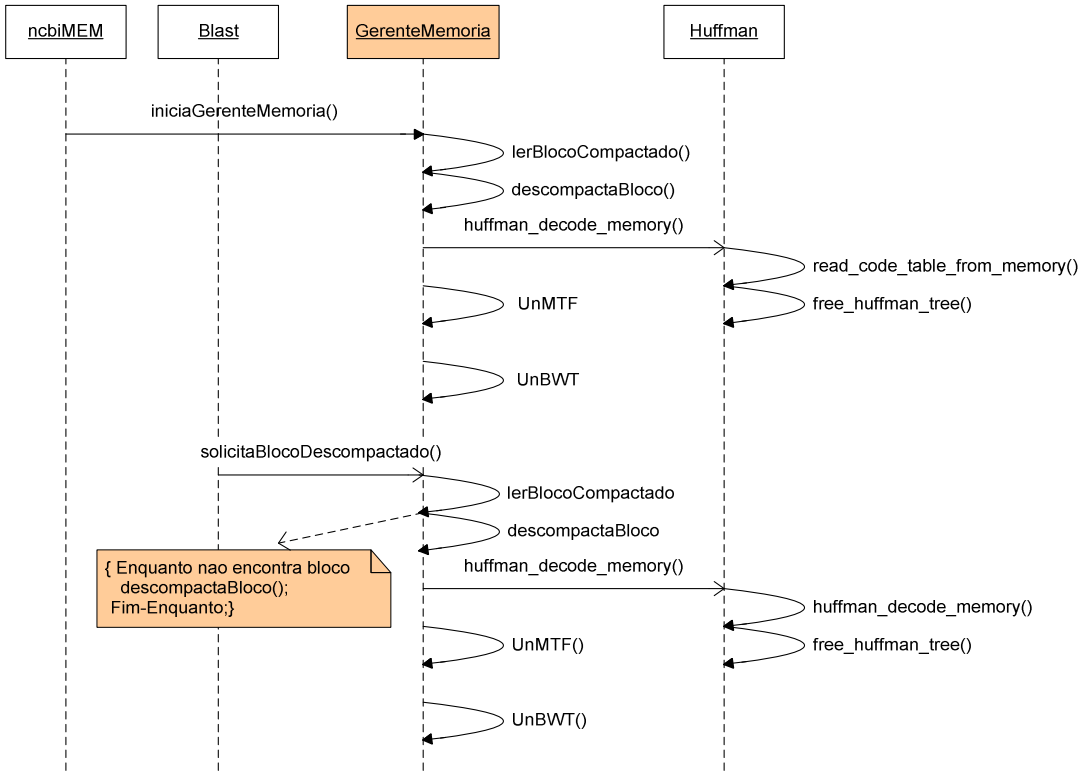

Figura 19 - Interação entre BLAST e Gerente de Memória

5.3 Conclusão

Neste capítulo foram discutidos os aspectos relacionados com a especificação de uma proposta para lidar com o problema persistência e acesso a dados de biosseqüências. Após uma breve discussão sobre a solução adotada, foram detalhados os componentes de implementação da presente proposta e a forma como estes componentes se organizam e se relacionam para atingir o objetivo funcional da aplicação.

Cada módulo do programa implementado foi descrito separadamente usando diagramas da UML. Esses diagramas foram utilizados tanto para a especificação estrutural do programa, através de diagramas de classes, como para a descrição dos aspectos dinâmicos do sistema.

Como próximo passo, serão apresentados no capítulo seguinte os resultados obtidos durante a avaliação prática da estratégia definida.