A FRAMEWORK-BASED OBJECT-ORIENTED DESIGN OF BLAST

GENG JIN

A THESIS

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

SEPTEMBER 2003

Your file Votre référence
ISBN: 0-612-83905-2
Our file Notre référence
ISBN: 0-612-83905-2

Abstract

A Framework-based Object-Oriented design of BLAST

Geng Jin

Sequence databases are growing rapidly in size, so there is a need for faster algorithms. This requires reuse of existing code for data structures, algorithms, and heuristics in order for rapid development of new algorithms. Blast is one of the most heavily used programs for searching sequence databases. The original Blast is a procedural program, not particularly well-suited for reuse. In this thesis, we develop an object-oriented framework to meet the requirement of reusability. First we describe the methodology for Framework Programming. Then we use the methodology to develop the Blast framework. Finally, we apply the framework to develop BlastP, an instance of Blast, in order to evaluate the Blast framework.

Acknowledgments

Nobody has helped my professional growth more than my supervisor, Dr. Greg Butler. He not only taught me much knowledge of bioinformatics and database, but also guided me into a new world of computer science. Without his guide, I could not accomplish this thesis.

I would like to thank my colleagues, Liusong Yang and Yang Li, whose knowledge influenced this work. I also thank Jian Sun who supported to test the BLAST system.

I would like to thank my friend and schoolmate Guangyi Wang. I got lots of benefit from every our discussion, and these directly promoted this thesis accomplish. I appreciated Ju Wang and Bin Nie whose knowledge and experience helped the development of the thesis.

I gratefully acknowledged the Department of Computer Science of Concordia University and Quebec government for all devices and financial support.

Finally, yet importantly, I deeply thank my dear wife, Yifang Zhao. Without her love and support, I could not go through my research at Concordia University.

Content

List of Figures

List of Tables

Acronyms

BLAST:	Basic Local Alignment Search Tool
DB:	Database
FP:	Framework Programming.
GP:	Generic Programming.
HSP:	High -score Segment Pair.
MSP:	Maximal Segment Pair
NCBI:	National Center for Biotechnology Information
PHI-BLAST	Pattern-Hit Initiated BLAST
PSI-BLAST	Position-Specific Iterated BLAST
RPS-BLAST	Reverse Position Specific BLAST
STL:	Standard Template Library.
UML:	Unified Modeling Language.
URL:	Uniform resource locator
WU-BLAST:	Washington University BLAST
XP:	Extreme Programming

Chapter 1

Introduction

Alignment search tools are the most important things for the bioinformatics world. A high-speed search tool would benefit biologists. However, to improve the speed of alignment is really a big challenge. BLAST [1] (Basic Local Alignment Search Tool) is one of many successful algorithms. However, most of BLAST programs lack reusability. It is difficult for programmers to maintain when algorithms update. We have tried to design a flexible framework to solve this problem.

1.1 Related Work

BLAST is a set of search tools to calculate similarity for biological sequences. The main reason biologists are interested in the similarity between sequences is that similar sequences hold a similar function, structure, and evolutionary history. A sequence alignment is a corresponding relationship of residues, gene characters, between two sequences.

Because sequence databases grow dramatically, faster algorithms are always required. Currently, BLAST has several different versions. One of the main versions is NCBI (National Center for Biotechnology Information) BLAST [2] that includes nucleotide BLAST, protein BLAST, translated BLAST search, and conserved domains search. The nucleotide BLAST has Blastn and MEGABLAST. Blastn is the standard BLAST program for nucleotides, and MEGABLAST is for the batch

nucleotide queries. Protein BLAST has three programs: Blastp, PSI-BLAST and PHI-BLAST. Blastp is the standard BLAST for protein, PSI-BLAST is for finding the members of a protein family or building a custom position-specific score matrix, and PHI-BLAST is for finding proteins similar to the query around a given pattern. Translated BLAST search includes Blastx, which does nucleotide query against protein DB, tBlastn, which does protein query against translated DB, and tBlastx, which does nucleotide query against translated DB. RPS-BLAST is for finding conserved domains in the query.

WU-BLAST [3] is another important version of BLAST, which was developed by Warren Gish at Washington University. WU-BLAST is also a powerful toolkit, but totally different from NCBI-BLAST. Also, BLAST has the distributed implementation, TurboBLAST [4], and the parallel implementation, mpiBLAST [5].

1.2 Problem Statement

Bioinformatics technology, both data and algorithm, is constantly advancing, but today's BLAST program cannot adapt to the situation. The primary problem is one of infrastructure, that is, BLAST lacks reusability. First, many versions of BLAST have their own infrastructure, data structures, and algorithms. Most of them are constitutionally different and cannot be used interchangeably. Second, in all but one version of BLAST, it is very difficult for programmers to synchronously maintain the system when data or algorithms are updated because Procedural Programming has an innate defect with respect to maintenance. Moreover, for an open source

projected maintained by Concurrent Versions System (CVS) [7], it is necessary to the BLAST program to have a clear structure. It is also impossible to "mix-and-match" components from several BLAST programs into one system.

The second problem is for using BLAST system. In the early age of using BLAST, biologists needed to download BLAST programs and install them into the local computer. Now, there are web servers for BLAST. However, users have to know a URL address of a BLAST server if they want to do a BLAST alignment. Although users may know the URL address of a BLAST server, there is still a bad thing: they may not know whether the BLAST service is available or not. When a BLAST developer maintains the BLAST server, the developer may not notice every BLAST user. Of course, the users cannot know when a new BLAST web server plugs into the web. Moreover, facing many BLAST servers, users cannot easily choose a satisfactory one.

In summary, the problem domain has two fields: easy of use and reusability. The application of BLAST is a dynamic network application, and the development of BLAST is dynamic too. Facing and interacting with this dynamic environment, both BLAST users and BLAST developers want to control it but not go oppositely.

1.3 The Idea and Goal

Our bioinformatics team tried to construct a dynamic model to overcome the said problems. First, we want to design a dynamic system platform to support a dynamic network application. Second, we employ object-oriented technology and component

technology to solve the problem of reusability.

1.3.1 Dynamic Network Application

The concept of network service satisfies the requirement of a dynamic system. A service discovery framework provides a collection of protocols for developing dynamic client/server applications, allowing clients to find and use services without any previous knowledge of the locations or characteristics of the services. There are currently many service discovery technologies available or under development, including Jini [6, 29,30], UPnP (Universal Plug and Play) [31], Salutation [32], SLP (Service Location Protocol) [33], Bluetooth SDP [34], and Ninja [35,36]. They employ different architectures to solve similar problems, which is to propose certain ways and means of device interaction with the ultimate aim of simple, seamless and scaleable device inter-operability.

Briefly, it is not easy for coordination frameworks to deal with lots of communication protocols between devices. A coordination framework can allow devices to know each other in detail, but also can define a standard protocol in public. The problem is how to balance between standardization requirements and device autonomy. Jini takes standardization to one extreme, while UPnP takes autonomy to another extreme. Salutation treads a middle way between autonomy and standardization.

Although Jini is only one competitor in a non-empty market, and what will condition the success or failure of Jini is partly the politics of the market, we

gingerly choose Jini according to that requirement – dynamic client / server service structure. Because Jini is a standardization of dynamic query / response system, it satisfies us more than the others. Moreover, Jini is open, and easy to implement. Though it relies on the Java Virtual Machine (JVM), the feather of "run everywhere" is attractive. Moreover, we need the technology of Java Native Interface (JNI) that allows Java to embed native language, since we use C++/STL to implement the BLAST framework, which should be called by Jini. Although UPnP naturally has inter-operability with many kinds of native languages, it is system-dependent, relying on Windows. In Salutation, Salutation Managers (SLM) [32] do not satisfy our requirement.

1.3.2 Object-Oriented Framework

For the reusability requirement, we tried to design a bioinformatics search tool framework [8].

1. Design a working and easily extendable framework. That means users can directly use our BLAST based on our framework, and can easily develop their own bioinformatics search tool by extending this framework.
2. According to the concept of Generic Programming (GP) [9], we separate data structures and algorithms, class declaration and implementation. All parts are independent. In this way, data and algorithm can be updated independently.
3. Construct the framework based on the Standard Template Library (STL) [10]

that is a well-done C++ general framework. This can help us quickly develop our framework.

4. Following GP, we developed a template function library and a container library as two components in our framework. This is the implementation of (1).
5. Design stable interfaces for every function.

1.4 BLAST Jini System

The Jini system is three-tier structure with client, lookup server, and service server. When a service server has a new service, it tells its lookup server some information about the service, e.g. the location of the service. A client only asks the lookup server when it needs a service, and it is not necessary for the client to know where the service is and how to run it. As an agent of the client and service servers, the lookup server manages the service information and responds to clients.

There are basically three processes in the workflow of the BLAST Jini System (see Figure 1). First, service registration: When a Blast server can provide a new BLAST service, it will apply for registration on the BLAST Locator (BLAST lookup server), and then, in response to a request for a BLAST server, the BLAST Locator will send a Registrar (a proxy of the BLAST Locator) to the BLAST sever (see Figure 1/(1)). The BLAST server gets the Registrar and sends the BLAST proxy to the BLAST Locator (see Figure 1/(2)). Second, service discovery: When a client has a requirement of a BLAST service, it will apply for lookup BLAST service in

the BLAST Locator, and the client will get Registrar from the BLAST Locator as response (see Figure 1/(3)). Then the client downloads a BLAST proxy from the BLAST Locator (see Figure 1/(4)). Finally, connect by RMI (Java Remote Method Invocation system): the client connects to the BLAST server by RMI (see Figure 1/(5)). The client sends a query to the BLAST engine, which performs the calculations for the search, and then the BLAST server responses.

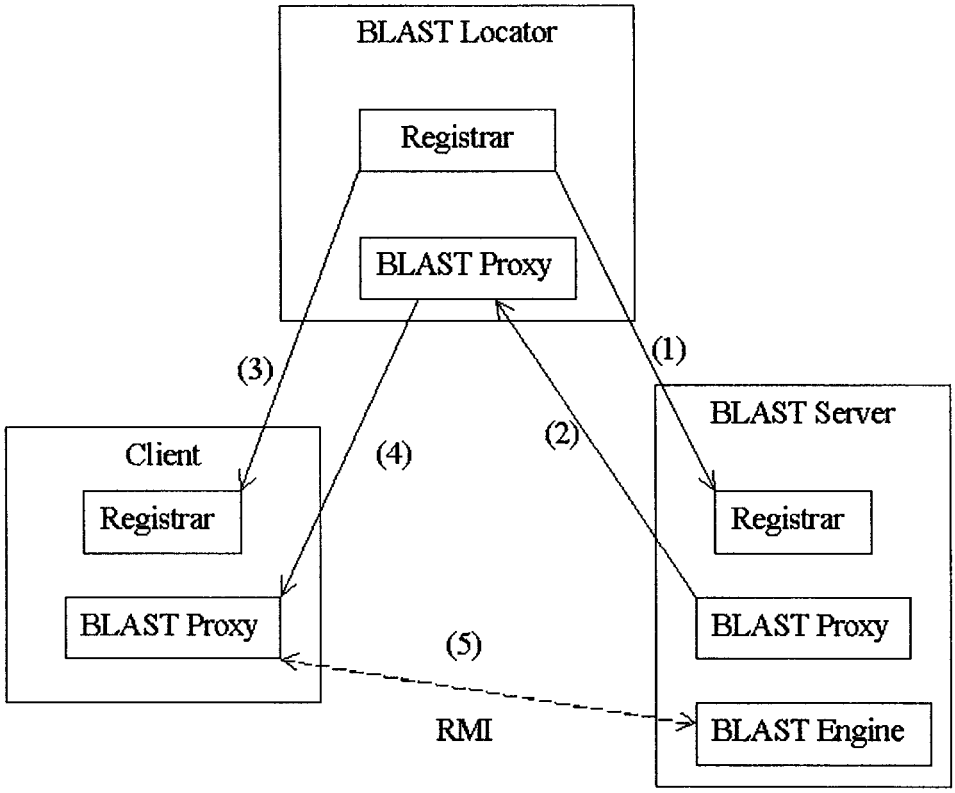

Figure 1 Process of BLAST Jini System

1.5 Contribution of the Thesis

Our team, according to the scope of design tasks, separates the whole system into two sub-systems:

- The sub-system of BLAST Jini system, and

- The sub-system of BLAST engine framework.

There are three parts in the project. My colleagues, Liusong Yang , whose thesis is "A Design of a BLAST Server with Jini Technology", designed the whole system; and Yang Li, whose thesis is "Workflow management system for Jini-based Blast cluster server", designed in detail and implemented the sub-system of BLAST Jini system. This thesis focuses on designing in detail and implementing the sub-system of BLAST engine framework, briefly called the BLAST framework.

1.6 Organization of the Thesis

This thesis has six chapters. Chapter 2 talks about the background of BLAST and the methodology we use. From chapter 3 to 6, our design is described in detail. Chapter 3 and 4 talk about the top-level and detailed design respectively. Chapter 5 describes examples of application and the cookbooks for BLAST developers. Finally, chapter 6 gives our conclusion and discusses future work.

Chapter 2

Background

Before our design, we have to fully understand the BLAST algorithm. BLAST is such a complex system that it cannot be discussed in one chapter in detail. This chapter only gives a system-level overview of BLAST and uses an example, Blastp, to describe the BLAST algorithm. This background of BLAST is taken from Shawn Delaney's master thesis [18].

In addition, we use Framework Programming (FP), Generic Programming (GP)[9,10] and Unified Modeling Language (UML), to guide the development and documentation our project.

2.1 BLAST Overview

The role of BLAST is the function of Database Searching in the following picture (Figure 2. -http://www.bmm.icnet.uk/people/rob/CCP11BBS/flowchart2.html). Biologists collect unknown biological sequences from experimental data, and then run BLAST to identify them. Generally, BLAST compares the query sequence against standard sequences one by one from a biology database that holds already-identified standard sequences. Three leading biology databases are GenBank [11], European Molecular Biology Laboratory (EMBL) [12] and DNA Data Bank of Japan (DDBJ) [13]. The BLAST comparison is not an exact match but an approximate match because single nucleotides or amino acids mutate during

evolution. A score matrix *S* is used in which scores represent the quantification of the mutation rate. For example, a score *S[a1,a2]* is a normalized frequency that amino acid *a1* mutates into *a2* over one unit of evolutionary time. When BLAST does an alignment between a query sequence and a subject sequence, it derives the score from a score matrix for each aligned amino acid pair of two sequences, and sums the score. If the sum score is higher than a given *Score Threshold*, some information of the subject sequence will be saved as result.

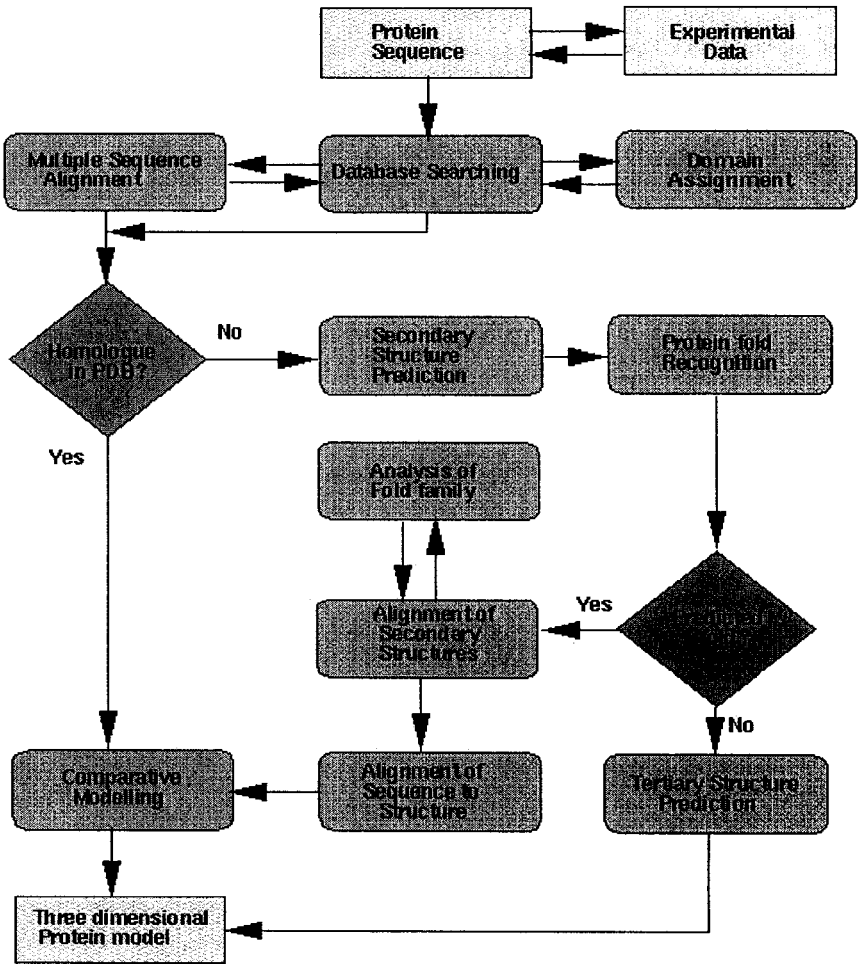

Figure 2 Role of Database Searching (BLAST) in Genome Analysis Workflow

2.1.1 Input and Output of BLAST

BLAST is designed for fast database scanning to look for similarities between a query protein sequence and those in a protein sequence database. The algorithm is heuristic in nature, allowing the user to modify the search sensitivity by assigning values to various parameters. By default, the program input is a query sequence in FASTA format (Figure 3).

```
> Yeast |>GCGSEQ.TMP Nature 387:87-90 [97313267] (1997)
PEVYILILPGFGMISHIISQESGKKETFGCLGMIYAMMAIGLLGFVVWAHHMFTVGMDID
TRAYFTSATMIIAVPTGIKIFSWLATLHGTQINYSPSMLWSLGFVFLFTVGGLTGVILAN
SSIDVTLHDTYYVVAHFHYVLSMGAVFAIMGGFVHWYPLFTGLTLNPYLLKIQFFTMFIG
VNLTFFPQHFLGLAGMPRRYSDYPDTYTSWNIISSLGSYISLIATMLMLMIIWESLINKR
IILFPLNMNSFIEWYQNLPPAEHSYNELPILSNF
```

Figure 3 An Example of Input Data of Query Amino-acid Sequence

BLASTP 2.2.1 [Apr-13-2001] RID: 1004243022-9073-16278

Query= GCGSEQ.TMP Nature 387:87-90 [97313267] (1997) (1048 letters)

Database: nr 787,946 sequences; 250,424,713 total letters

Figure 4 Example BLASTP Output - Program Introduction

The program implements two tasks: (1) It scans the sequence database with an input query sequence and compiles a list of *HSPs(High Score Pairs)*, and (2) analyses the *HSP* list in order to assign statistical significance to those matches. For example, the output of Blastp of NCBI has five parts: (1) Program introduction (Figure 4), (2) Histogram of expectations if one is requested, (3) List of one-line

summaries for each matching database sequence (Figure 5.), (4) List of *HSPs(High Score Pairs)*(Figure 6.), and (5) Parameters used and search statistics. Parts (3) and (4) are of general interest to users and are further described here.

Sequences producing significant alignments:	Score (bits)	E Value
gi|539355|pir||S14177 SCD25 protein (version 1) - yeast (Sa...	1791	0.0
gi|2133118|pir||S64758 SCD25 protein (version 2) - yeast (S...	1842	0.0
gi|1351042|sp|P14771|SC25_YEAST SCD25 PROTEIN >gi|457494|gb...	1835	0.0
gi|6323341|ref|NP_013413.1| cell division cycle blocked at ...	566	e-160
gi|3484|emb|CAA27259.1| (X03579) CDC25 protein (aa 1-1588) ...	550	e-155

Figure 5 Example BLASTP Output - One Line Summary

```
Score = 47.8 bits (112), Expect = 5e-04
Identities = 31/120 (25%), Positives = 53/120 (43%), Gaps = 6/120 (5%)
Query: 3   ITSSPDLFYLNDCDVVYWYDLTRLVCHYVNLTERDLLANEREKFLTSLDLLTAQITYVYM 62
           I +  D+FY    D+  W +L  L  +Y     +  L   R  F    DL++  I +  +
Sbjct: 443 ILAKSDIFYHYSRDIKLWTELQDLTVYYTKTAHKMFLKENRLNFTKYFDLISDSIVFTQL 502

Query: 63  LFRNL------RLVEDSFKKTLKKLIYTLSRFSINANIWFHSTLFEEREAIASQKDPERR 116
             R +      +      KK  K LI +LSR SIN++++F S    ++    + KD + +

Sbjct: 503 GCRLMQHEIKAKSCSKEIKKIFKGLISSLSRISINSHLYFDSAFHRKKMDTMNDKDNDNQ 562
```

Figure 6 Example BLASTP Output - List of HSP

The one-line summary list facilitates the comparison of the scores of individual matches. The first column, Sequences producing significant alignments, is the sequence descriptor from the FASTA format, and contains the sequence identifier and name. The second column, Score, contains the score of the highest scoring *HSP* — the *MSP(maximal segment pair).* The query may have more than one *HSP* with a

subject, but only the highest scoring one is reported in the one-line summary. The third column, Expect value (E), is a parameter that describes the number of hits one can "expect" to see just by chance when searching a database of a particular size.[14]

The list of *HSP* has the statistical summary and the alignment of the query segment and the subject segment. The offsets of the *HSPs* are placed at the beginning and the end of the query and subject segments. Between the query and subject, letters indicate exact matches, while "+" indicates a non-identical but positive scoring match. No symbol indicates a zero or negative score for that residue pair, while "–" shows the gap.

2.1.2 BLAST Algorithm

Dynamic programming tends to break the original problem to sub-problems and chooses the best solution in the sub-problems, beginning with smaller in size. The best solution in the bigger sub-problems is found by using the best ones of the smaller sub-problems; and finally, dynamic programming uses a retroactive way to connect all sub-solutions from bottom to top. The basic idea behind dynamic programming is the organization of work in order to avoid the repetition of already completed work.

The first milestone of applying dynamic programming in bioinformatics sequence comparison is the algorithm of Needleman-Wunsch[15] and Smith-Waterman[16], which are rigorous dynamic programming algorithms for

calculating the optimal global alignment and local or sub alignment. A global alignment is a match between the complete sequences; that is, to find the best alignment over the entire sequence lengths, whereas a local alignment is a match between segments of each sequence. Global alignment is often chosen when two sequences are known closely-related, while local alignment can search distantly-related sequences because it is good at sub-segment similar search. Therefore, for unknown sequences, local alignment is more meaningful for biologists.

The time complexity of Needleman-Wunsch and Smith-Waterman algorithms is $O(n \times m)$, where n and m are the lengths of the two sequences. When we are doing a database search, the time complexity is $O(k \times n \times m)$, where k is the size of database. That means the dynamic programming algorithm is hard to apply to search large databases, therefore, heuristic algorithm is developed to improve computing speed. The most popular heuristic algorithms are FASTA[17] and BLAST[1] (Basic Local Alignment Search Tool). FASTA has two steps: first, word search, to find highly similar segments in the two sequences; second, working on the result of word search, calling Smith-Waterman alignment to extend within a window. BLAST is a three-step algorithm [18]: building a neighborhood, a word search called hit detection, and extension of the hit. The neighborhood $N(V)$ of a word V of length W in the query is the set of words of the same length W that are highly similar to V. That is, a neighbour is a mutation of V that is common or probable according to the statistical matrix S. the neighborhood N of the query is the amount of $N(V)$ for all V

in the query.

The heuristic algorithm can reduce time complexity because it first fills a sub-set of entries that form a highly similar segment, and then neighboring entries are filled until an extension is maximum.

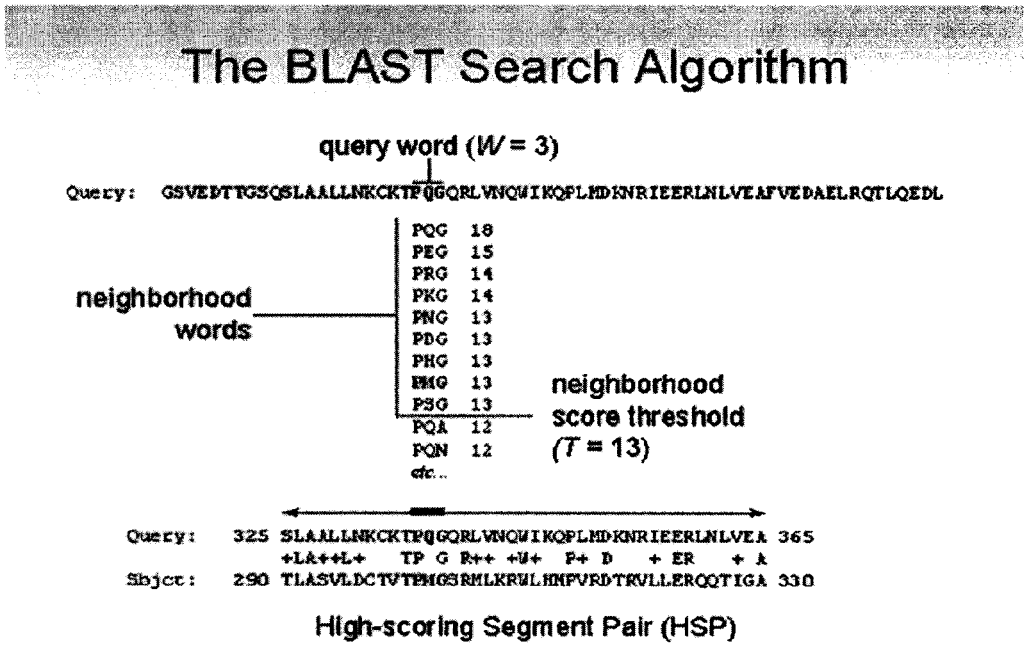

Figure 7 The BLAST Search Algorithm

2.1.3 BLAST Workflow

BLAST follows a similar scheme in that it relies on a core similarity, although with less emphasis on the occurrence of exact matches (see Figure 7[2]). This program also aims at identifying core similarities for later extension. The core similarity is defined by a window with a certain match density on DNA or with an amino acid similarity score above some threshold for proteins. Independent of the exact definition of the core similarity, BLAST rests on the precomputation of all strings

which are in the given sense similar to a position in the query. The resulting list may be on the order of thousand or more words long, each of which if detected in a database give rise to a core similarity. In BLAST nomenclature this set of strings is called the neighborhood of the query. The code to generate this neighborhood is in fact exceedingly fast.

The neighborhood is represented by finite automaton that is used to detect occurrences in the database of any string from the neighborhood. This automaton is a program, constructed on the fly and specifically for the particular word neighborhood that has been computed for a query. When scanning a database of sequences, the automaton is given an additional letter at a time and decides whether the string that ends in this letter is part of the neighbor. If so, BLAST attempts to extend rests of the similarity around the hit and if this is successful reports a match. In the following paragraphs, the workflow of BLAST is described in detail, using Blastp as an example.

1. **Neighborhood Construction** (Figure 8.) computes and outputs the neighborhood *N*. The query sequence *Q* is scanned word *V* by word, length *W*, and *N(V)* is computed by taking all words *U* of length W where *S[U, V]* scores at least *T*. Each word *V* may have zero or more neighbors. The set of neighbors of all query words is the neighborhood *N*.
2. **Hit Detection** (Figure 9). Each subject sequence *SB* in the database *DB* is scanned for exact matches to a word in neighborhood *N*.
3. **Hit Extension** (Figures 10,11). The hit *H*, is extended into a potentially

higher scoring alignment.

In Neighborhood Construction (see Figure 8), the query sequence Q is scanned from i = 0 to i = L-W+1. Every word of Q, $q_i\, q_{i+1} \ldots q_{i+w-1}$, has to match a assembled residues of AA, $n_0\, n_1\, n_2 \ldots n_{w-1}$, where the score, $M(q_i\, q_{i+1} \ldots q_{i+w-1} \| n_0\, n_1\, n_2 \ldots n_{w-1})$, not less than T. so that assembled residues of AA, $n_0 n_1 n_2 \ldots n_{w-1}$, is the neighbor of the word of Q, $q_i q_{i+1} \ldots q_{i+w-1}$. All the neighbors of all words of Q compose the output result, neighborhood N.

```
Q: query sequence, q0q1 q2...qL, size = L
M: function that return the alignment score
W: word size
T: threshold word score
AA: amino acid alphabet, size=K
N: neighborhood word set
<neighbor, offset>: a set of tuples of neighborhood, neighbor is the word,
n0n1 n2...nw-1, that matched the query word at offset, qi+w-1

N = Build-Neighborhood(Q,M,W,T)
begin
for (i = 0; i < L-W+1; i ++)
        if (∃ a word n0n1 n2...nw-1, where nj∈ AA,
            such that M(qiqi+1...qi+w-1|| n0n1n2...nw-1)> = T)
        then   N∪< n0n1n2...nw-1, qi+w-1>
end
```

Figure 8 Step 1 - Neighborhood Construction

The hit detection step (see Figure 9) scans every subject sequences in database. Each subject *SB* is scanned for exact matches to a member of the neighborhood *N*. When a match is found, the extension step is invoked a word hit *H* is an alignment of a query word and subject word whose offsets are *q_off* and *s_off* respectively.

```
DB: subject sequence database, size =dbsize
Q: query sequence
SB: subject sequence, size=m
N: neighborhood word set, <neighbor, offset>
H: word hit, composed of the offsets H on S and Q, (s_off, q_off)

hit(DB, N)
begin
    for(i = 0; i < dbsize; i ++)
        SB= DB[I]
        for (j = 0; j < m-W+1; j ++ )
            if (SjSj+1... Sj+W-1) ∈ N
            then H.s_off=Sj+W-1
                while (offset not end), offset point to the offset position list of Q
                    H.q_off=offset
                    Extend(H)
end
```

Figure 9 Step 2 - Hit Detection

The hit extension step (see Figures 10,11) is parameterized by the word hit *H*, the word size *W*, a scoring function *M*, the falloff score *X*, the threshold alignment score *S*, and the query and subject sequences *Q* and *SB* (1). This step attempts to extend a hit into a longer, potentially higher scoring alignment. The offsets *q_beg*, *q_end*, *s_beg* and *s_ end* mark the maximal scoring alignment. They are first set to the delimiters of the word hit (2). The first loop (3–8) then sets them to the delimiters of the maximal scoring sub-alignment within *H*. The hit is traversed from *q* to *q_pos*.

Step Three - Hit Extension

```
1 Extend (Q,SB,W,H,M,X,S)
2    q_beg = q = H.q_off-W, q_end = q_pos = H.q_off, s_beg = s = H.s_off-W, s_end = H.s_off
3    do
4       sum += M(Q_q||SB_s)
5       if(sum > score) then score = sum, q_end = q, s_end = s
6       else if(sum <= 0) then sum = 0, q_beg = q, s_beg = s
7       q++, s++
8    while(q < q_pos)
9    if((x = -score) < X) then x = X
10   leftq = q_beg, lefts = s_beg, rightq = q_end, rights = s_end
11   leftsum = rightsum = leftscore = rightscore = 0
12   Left Extension:
13   q = leftq , s = lefts, sum = leftsum
14   do
15      q--, s--
16      sum += M(Q_q||SB_s)
17      if(sum > 0) then
18         score += sum, sum = 0, q_beg = q, s_beg = s
```

Figure 10 Step 3 - Hit Extension

Residue pair scores are accumulated in *sum* (4). When *sum* is positive (5) it is added to *score* and *q_ end* || *s_ end* is advanced right. A negative *sum* (6) causes *q_beg* || *s_beg* to be advanced right effectively excluding the negative scoring residue pair from the maximal scoring sub-alignment of *H*. Parameters specific to either the left or right extensions are initialized in (10-11). The second loop (13–21) extends in the left direction. Residue pair scores are accumulated in *sum* (16). If *sum* is positive, it is added to *score* then reset to zero and the alignment is extended (17-18). If *sum* falls below *x*, the extension terminates (20). Longer extensions are favored by

allowing x to be set to *-score* (9,19,29). A right extension occurs if the conditions in (21) hold (which is always the case at first). The values of *s*, *sum* and *score* are saved in the left extension parameters, *lefts*, *leftsum* and *leftscore* respectively (22). The third loop works in the same manner except that the extension proceeds in the right direction (23–31). The left extension may continue if the conditions in (33) hold. If so, *s*, *sum* and *score* are saved in the right extension parameters, *rights*, *rightsum* and *rightscore* respectively (34). The maximal scoring alignment is stored in the *HSP* set if *score* meets the threshold *S* (36).

```
19          if((x = -score) < X) then x = X
20    while (sum >= x)
21    if (score > rightscore) ∧ (rightsum > X) ∧ (-rightscore > X) then
22          leftq = q, lefts = s, leftsum = sum, leftscore = score;
23          Right Extension:
24          q = rightq, s = rights, sum = rightsum
25          do
26              sum += M(Q_q||SB_s)
27              if(sum > 0) then
28                  score += sum, sum = 0, q_end = q, s_end = s
29                  if((x = -score) < X) then x = X
30              q++, s++
31          while (sum >= x)
32          rightq = q
33          if (score > leftscore) ∧ (leftsum > X) ∧ (-leftscore > X) then
34              rights = s, rightsum = sum, rightscore = score
35              goto Left Extension
36    if (score >= S) then HSP ⋃ ((Q_{q_beg}...Q_{q_end}||SB_{s_beg}...SB_{s_end})
```

Figure 11 Step 3 - Hit Extension (cont.)

Figure 12 shows dynamics of the extension algorithm. M is a score function. A word hit H is a sub-alignment of length W, H = (*q_beg...q_end* || *s_beg...s_end*)

whose score, $M(H) >= T$. The extension algorithm finds the locally maximal alignment starting from H. Extension continues in either direction until $M(q\_left...q\_beg' \parallel s\_left...s\_beg') < X$ or $M(q\_end'...q\_right \parallel s\_end'...s\_right) < X$. The total score of the alignment is $M(q\_beg'...q\_end' \parallel s\_beg'...s\_end')$

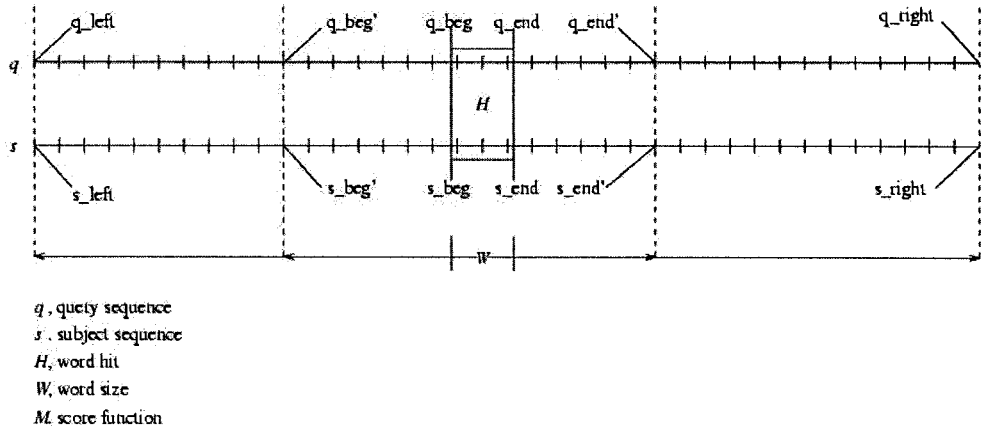

Figure 12 Dynamics of Extension Algorithm

Figure 13 shows the dynamic of extension score. The alignment extends until the sum of any direction extension, sum < X. The middle alignment contains the score of the highest scoring segment- the MSP, maximal segment pair.

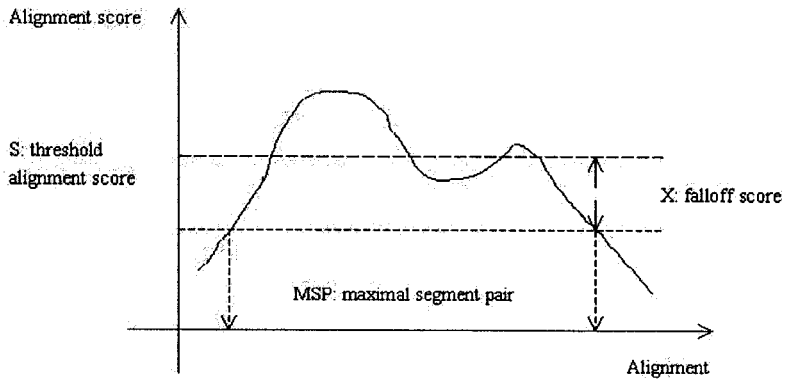

Figure 13 Dynamic of Extension Score

2.2 Methodology

We choose a framework-based object-oriented method to develop the BLAST framework to satisfy the requirement of reusability [8]. We name our methodology, Framework Programming(FP), which is a hybrid of eXtreme Programming(XP) [19]. We use UML [20] to model our design. We follow the principle of Generic Programming (GP)[9] that separates data structures and algorithms to design the framework, and we use STL [10], an example of the GP technique, to implement the framework.

Butler said[21] reuse applies not only to source-code fragments, but to all the intermediate products generated during software development, including requirements, documents, system specifications, and design: indeed any information that the developer needs to create software. It is important to separate the two sides of the reuse process: development of components for reuse and development of new applications by reusing of existing components. The first side produces reusable components, and the other side consumes them during the development of new applications. It is important to state that while most reuse may be the reuse of source code, the term component also includes designs, specifications, and requirements.

2.2.1 Framework

A framework is a reusable design expressed as a set of abstract classes and the way their instances collaborate. It is a reusable design for all or part of a software system. A framework supports the development of a family of applications, which allows the

user to reuse abstract designs and pre-fabricated classes in order to develop a system in the domain. A user may also customize existing classes.

The design of the framework incorporates decisions about the distribution of control and responsibility, the protocols followed by components when communicating, and implementation for each of the major algorithms. Often the implementations are template methods that embody the overall structure of a computation and that call user's classes to perform sub-steps of the algorithm. Default implementations of each user class may be provided and the user will subclass in order to override or specialize the operation that implements the sub-step.

2.2.1.1 Documenting Framework

Butler summarized [22] various styles of documentation for a framework, including examples, cookbooks and recipes, contracts, design pattern, framework overview, reference manual, design notebooks, and others. To document a framework to assist developers, he concludes what we need a framework overview, examples of framework application specifically designed as documentation tools, and a recipe cookbook.

2.2.1.2 Developing Framework

Butler [25,39] classified the approaches to framework development as

- example-driven, bottom-up, incremental development;
- top-down domain engineering;

- hot spot generalization;
- use case-driven; and
- hybrid, which combines one or more of the above.

These classifications are not mutually exclusive.

The classic development approach in the object-oriented community is example-driven, bottom-up, and incremental. A framework is refactored when new requirements are encountered, and the applications are developed one by one.

Step 1: Build first application

Step 2: Iterate

Step 2.1 Change impact analysis for n-th application

Step 2.2 Refactor existing framework to accommodate changes

Step 2.3 Build n-th application

Domain engineering often invests more effort up-front in the modeling of the domain and the specification of architecture for the product line. Their understanding of the domain is more complete, and is captured in range of models, including the context, scope, taxonomy, data dictionary, feature model, domain specific software architecture, descriptions of major functionality as use cases and algorithms, and exemplar systems in the domain. The process summaries as follow:

Step 1: Domain analysis

Step 2: Develop domain specific software architecture (DSSA)

Step 3: Implement DSSA

Step 4: Populate DSSA as applications required

Hot spot generalization identifies the points of flexibility — the *hot spots* —required in the product line, classifies the type of flexibility required (using patterns or meta-patterns), and then assigns a subsystem with a facade class within the design to implement the hot spot.

Of course, hybrid top-down and bottom-up approaches are possible, where one considers the development of a few applications at once and performs a partial domain analysis. Interleave means these three steps can be done in parallel, resulting in their substps being interleaved.

Step 1: Interleave

Step 1.1 Partial domain analysis

Step 1.2 Change impact analysis and refactoring

Step 1.3 Build n-th application

2.2.2 Framework Programming (FP)

Framework Programming (FP) is a hybrid approach. It combines the modeling aspects of the top-down domain engineering approaches, and the iterative, refactoring approaches of the bottom-up object-oriented community. It employs domain analysis, pattern-driven, the prototype approach of use case-driven. In addition, it derives the features of test-driven and people-oriented approaches from XP.

2.2.2.1 Background of XP

Agile methodologies are new methods, which attempt a useful compromise between

no process and too much process, providing just enough process to gain a reasonable payoff. Agile methodologies are less document-oriented, usually emphasizing a smaller amount of documentation for a given task. In many ways, they are rather code-oriented: following a route that says that the key part of documentation is source code.

M. Fowler [24] believes that agile methods have the feature of the lack of documentation is a symptom of two much deeper differences: Agile methods are adaptive rather than predictive; agile methods are people-oriented rather than process-oriented. It is very difficult to see what value a software feature has until we use it for real. The problem with a project is that the requirements are always changing. Fowler thinks the agile methods welcome change, and it makes software to be *soft*. He said we need an honest feedback mechanism, which can accurately tell us what the situation is at frequent intervals. The key to this feedback is iterative development.

Of all the agile methodologies, XP [19] is the most prominent; Kent Beck is the designer. XP begins with four values: Communication, Feedback, Simplicity, and Courage, which create a new development team model – Pair Programming [40]. It then builds up to a dozen practices that XP projects should follow. Many of these practices are old, tried and tested techniques, yet often forgotten by many, including most planned processes. As well as resurrecting these techniques, XP weaves them into a synergistic whole where each one is reinforced by the others.

XP puts testing at the foundation of development, with every programmer

writing tests as they write their production code. The tests are integrated into a continuous integration and build process which yields a highly stable platform for future development.

XP builds an evolutionary design process that relies on refactoring a simple base system with every iteration. All design is centered on the current iteration with no design done for anticipated future needs. The result is a design process that is disciplined, yet startling, combining discipline with adaptivity in a way that arguably makes it the best developed of all the adaptive methodologies.

The key benefit of XP for the customer is a much more responsive software development. A usable, although minimal, system can go into production early on. The customer can then change its capabilities according to changes in the business, and also from learning from how the system is used in reality.

2.2.2.2 Framework Programming (FP)

Figure 14 shows the processes of FP.

Step 1 iteration

1.1 iteration

1.1.1 build or refactor the prototype

1.1.2 test

1.2 iteration

1.2.1 abstract or refactor hooks and hot spots

1.2.2 test

1.3 test, go back 1.1 or 1.2

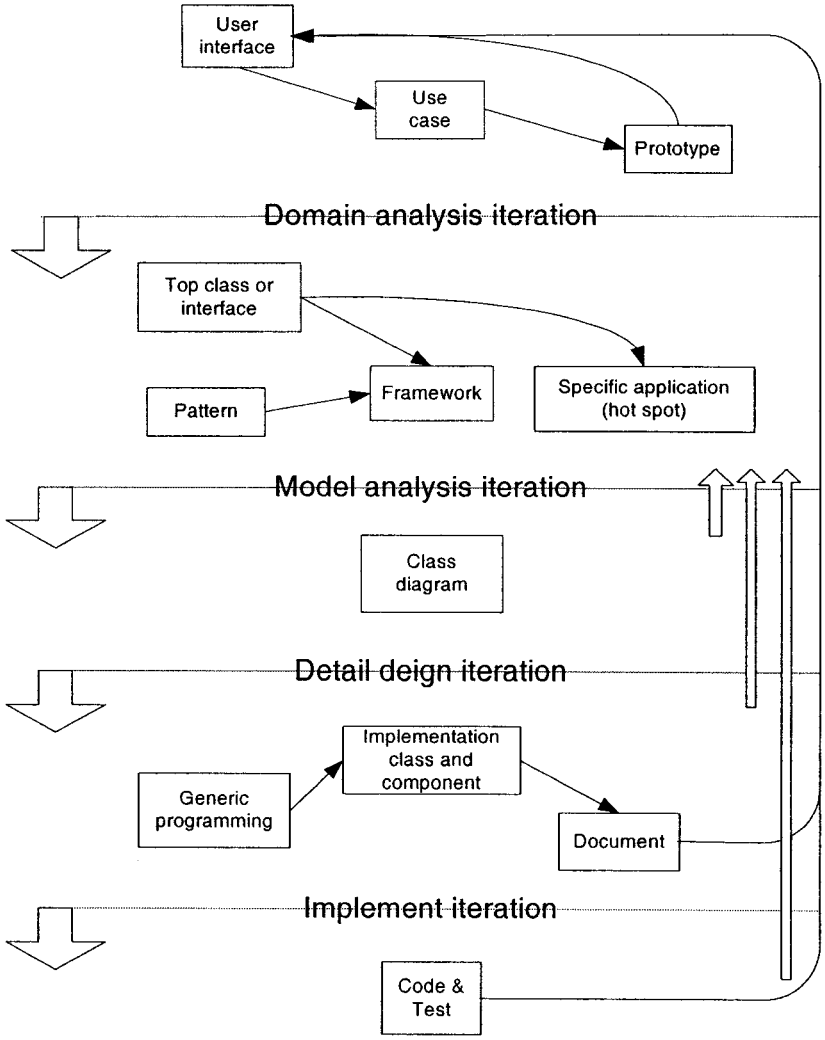

Figure 14 The Model of FP

We can see that customers take part in every loop of the development. Iteration 1.1 is driven by the customer aided by the programmer. In iteration 1.2 programmers need the feedback from the customer to prove that the work goes right. Of course, according to the feedback from the programmer, the customer has to adjust the prototype of the system (1.3 test). To abstract or refactor the framework and the hot spot, there is a five-step iteration [8]:

1. categorize the problem
2. research similar problem
3. prepare a domain description
4. analysis generic facts of the domain and separate the specific ones into the fold of hot spot.
5. repeatedly review and refine the analysis

In FP, there are five stages: domain analysis, model analysis, detail design, implement, code and test. The difference from the classical Object-Oriented methodology is that FP goes fast. It only does partial domain analysis to abstract the function and iterate. The time of iteration is about 1-3 weeks. In domain analysis, the use case approach is employed, which is the simplest and fastest tool for developing prototype. In modeling the framework, a pattern-driven approach helps the programmer to save a lot of time. GP can create the implementation of the design efficiently. The thing is that the process should go back whenever a bug is found.

2.2.2.3 Model of Development Team of FP

FP is one kind of agile method. It is suitable for small teams, and it is designed for the developer who does not have experience. FP agrees with the principle of people-oriented [24] rule of agile method. These people are customers and programmers. FP believes that the customer is the controller and the programmer is the generator.

FP believes that the requirement is a test and the test is the software. Hence, the customer builds the requirement, that is, the customer builds the software (regarding the programmer as a tool). FP believes customers have the right and ability to develop their own system, and they also have the ability to present their expectation or ideas into the system which they finally want, aided by the programmers.

FP welcomes customers. The Figure 15 shows the model of the developing team of FP. FP believes that the customer, expert user or professional operator, should be one of developing team members, whose duty is to approve or test the project. The model for programmers' working is Pair Programming [40] derived from XP, and the model for developers and user should be triangle programming.

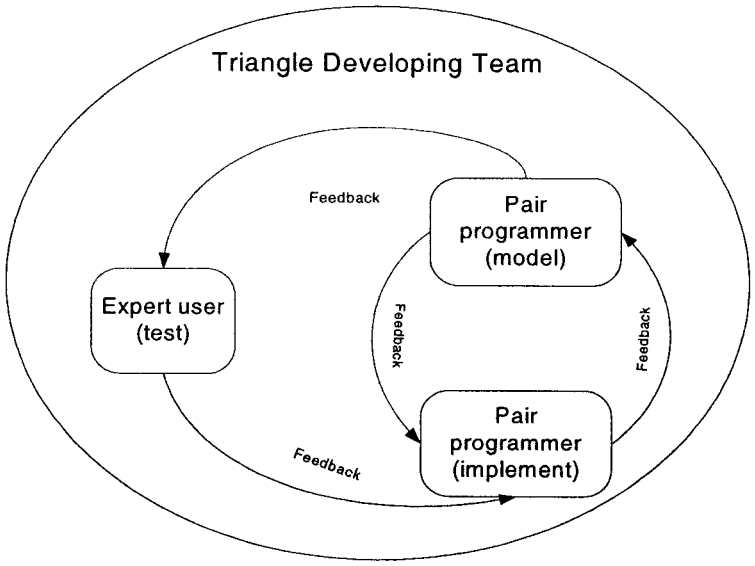

Figure 15 The Model of Development Team of FP

2.2.3 Generic Programming(GP)

Musser's working definition of Generic Programming [9,10] is "programming with concepts," where a concept is defined as a family of abstractions that are all related

by a common set of requirements.

GP is a discipline that studies the systematic organization of useful software components. The objectives are to develop a taxonomy of algorithms, data structures, memory allocation mechanisms, and other software artifacts in a way that allows the highest level of reuse, modularity, and usability. The purpose of GP has been to explore methods of developing and organizing libraries of generic or reusable software components. The meaning of reusable is "widely adaptable but still efficient" [9].

The Standard Template Library (STL) [10] is the embodiment of years of research on GP. STL is a general-purpose library of generic algorithms and data structures. It makes a programmer more productive in two ways: first, it contains a lot of different components that can be plugged together and used in an application, and more importantly, it provides a framework into which different programming problems can be decomposed.

The framework defined by STL is quite simple: two of its most fundamental dimensions are algorithm and data structure. The reason that data structures and algorithms work together seamlessly is the fact that they do not know anything about each other.

STL is designed with four fundamental ideas in mind: Generic programming, abstractness without loss of efficiency, von Neumann computational model, and value semantics[10]. The importance of STL lies more in its concepts than in the actual code or the details of its interfaces.

In fact, the BLAST framework regards GP as a pattern [21,23] to organize the data structures and algorithms and utilities STL to implement the framework.

2.2.4 UML

The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of the best engineering practices that have proven successful in the modeling of large and complex systems [26,27].

Chapter 3

Framework Overview

The document of our design begins with this chapter. This chapter gives an overview of the BLAST framework including the analysis of the domain problem and possible solutions. In our design, we assume that the reader is familiar with the UML and with the features of the ANSI standard C++ program language.

3.1 System Context

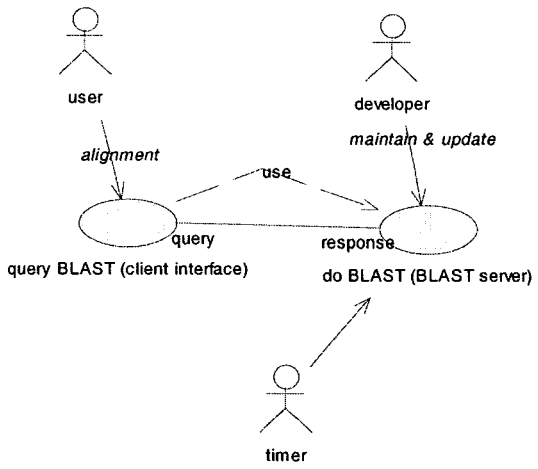

Figure 16 Use Case of A BLAST Web System

The BLAST System is a query/response system, which is waiting for the input of a query sequence, and then to align it using some other resource, for example, databases and score matrixes. Consider a BLAST web system (see Figure 16), a biologist submits an online query, and the system responds with the similarity result. Moreover, bioinformatics developers need to maintain or update the new parts and

features of the system. Tables 1,2 and 3 show the scenarios of BLAST web system.

3.1.1 Scenario of Query BLAST

A BLAST user does an alignment using a BLAST web server. The scenario is described as follows:

1. A user opens a web browser and inputs a URL of BLAST server.
2. The browser connects with the BLAST server and downloads the interface of BLAST server.
3. The user inputs a query and sets up the parameters of the BLAST program into the interface.
4. The user clicks the button of "Submit" in the interface to begin the query task.
5. The BLAST client interface sends the query task to the BLAST sever and waits for a response.
6. The interface receives the response and displays it to the user.
7. If the user has a new query task, the user goes back to (3).
8. The user exits the BLAST client interface and disconnects from the BLAST server.

Table 1 Scenario of Query BLAST

Use Case Name	Query BLAST
ID	Table 1.
Reference ID	Chapter 3.
Actor(s)	User
Pre-condition	User has to know the URL of BLAST server.
Normal Course	1. A user opens a web browser and inputs a URL of BLAST server. 2. The browser connects with the BLAST server and

	downloads the interface of BLAST server. 3. The user inputs a query and sets up the parameters of the BLAST program into the interface. 4. The user clicks the button of "submit" in the interface to begin the query task. 5. The BLAST client interface sends the query task to the BLAST sever and waits for a response. 6. The interface receives the response and displays it to the user. 7. If the user has a new query task, the user goes back to (3). 8. The user exits the BLAST client interface and disconnects from the BLAST server.
Alternate Course 1	(3) If the input sequence is not "good", the interface will warn the user to check and re-input. The warning process will stop when the user inputs a good query sequence, and move onto next process (4).
Alternate Course 2	If the user wants to cancel the task, click "cancel" button or "X" button in the right-up window, the interface window will close and disconnects from the BLAST server.
Post-condition	Null
Related Use Cases	This use case needs to call the use case of BLAST server.

3.1.2 Scenario of Do BLAST—Timer Behavior

When a BLAST server receives a BLAST task, it responds and sends back the result. The scenario is described as follows:

1. The BLAST server gets a query from a client, sets up a connection, and sends an interface of itself to the client.
2. The BLAST server gets a query task from the interface.
3. Depending on the parameters of the query task, the BLAST server chooses a set of algorithm, database and score matrix to construct the alignments.
4. The BLAST server returns the result to the interface.

5. If there is another query task from the interface, it goes back to (2).

6. The interface is disconnected from the BLAST server.

7. The BLAST server is waiting for another query from web.

Table 2 Scenario of Do BLAST - Timer Behavior

Use Case Name	Do BLAST
ID	Table 2
Reference ID	Chapter 3
Actor(s)	Timer
Pre-condition	A BLAST server is waiting for a query task from web
Normal Course	1. The BLAST server gets a query from a client, sets up a connection, and sends an interface of itself to the client. 2. The BLAST server gets a query task from the interface. 3. Depending on the parameters of the query task, the BLAST server chooses a set of algorithm, database and score matrix to construct the alignments. 4. The BLAST server returns the result to the interface. 5. If there is another query task from the interface, it goes back to (2). 6. The interface is disconnected from the BLAST server. 7. The BLAST server is waiting for another query from web.
Alternate Course	Null
Post-condition	Null
Related Use Cases	This use case is waiting for a call from the use case of client.

3.1.3 Scenario of Do BLAST—Developer Behavior

A BLAST developer updates a BLAST web server. The scenario is described as follows:

1. A developer shuts down a BLAST server.

2. The developer replaces an old BLAST program with a new one.

3. The developer restarts the BLAST server.

Table 3 Scenario of Do BLAST - Developer Behavior

Use Case Name	Do Blast
ID	Table 3
Reference ID	Chapter 3
Actor(s)	Developer
Pre-condition	Null
Normal Course	1. A developer shutdowns a BLAST server. 2. The developer replaces an old BLAST program with a new one. 3. The developer restarts the BLAST server.
Alternate Course	(1) If there are some tasks running on this server, they will terminate. A failure message will be sent to those clients.
Post-condition	Null
Related Use Cases	

3.2 User Characteristics

There are three kinds of actors in the problem domain: BLAST user, BLAST developer and framework developer. We abstract BLAST user from the perspective of the application; we abstract BLAST developer from the view of developing the BLAST program; we abstract BLAST framework developer from the view of developing reusable BLAST software.

BLAST user: utilize BLAST application. They commonly do BLAST alignment on a BLAST server, for example, biologists.

BLAST developer: develop, maintain, and update of BLAST program, including BLAST algorithm and data structure. They focus on some one specific BLAST system. They support the BLAST user.

BLAST Framework developer: develop components of BLAST program and maintain BLAST framework. They care about all BLAST application programs. They support the BLAST developer.

3.3 User Requirement

According to the scope of this thesis, here we only present the perspectives of BLAST developer and framework developer.

BLAST developer:

1. It is not necessary to shut down a BLAST system to maintain.
2. Easily maintain a BLAST system
3. Easily use shared components.

BLAST framework developer:

1. Easily develop shared components of BLAST program
2. Easily plug the shared components into a BLAST framework
3. Easily extend or maintain a BLAST framework

3.4 Possible Solution

According to the user requirement, here presents the possible solution here.

3.4.1 Scope

We decide to design an Object-Oriented BLAST Framework to accomplish the reusability of BLAST software. The features of Object-Oriented framework allow component plug-and-play, and allow ease of extension as well.

The goal of our design is satisfying all users: BLAST users, BLAST developers and BLAST framework developers. We choose User-Centered Software Design [41,42], which can guide us to reach the goal of our design.

The scope does not include:

1. BLAST Jini System is outside the scope of this thesis.
2. The requirement of BLAST developer 's (1), "it is not necessary to shut down a BLAST system to maintain", and BLAST framework developer 's (1), "easily develop shared components of BLAST program", is out of the scope of our design.

3.4.2 Possible Solution

Name: BLAST Framework

Type: Library framework

Methodology: Framework Programming (FP)

Implementation principle: We follow the principle of Generic Programming to separate data structures and algorithms in the BLAST framework.

Language: We choose C++/STL to implement the BLAST framework because their features are the best to support the development of Object-Oriented Framework.

Operating system (OS): We choose UNIX because it is the most popular system platform in the field of high-level server. Moreover, the application is easy to be translated from UNIX to LINUX.

Foundation and relationship: The relationship between framework components is the hierarchy structure in Figure 17. BLAST Framework derives from Bioinformatics search engine framework, and both of them use STL Framework components.

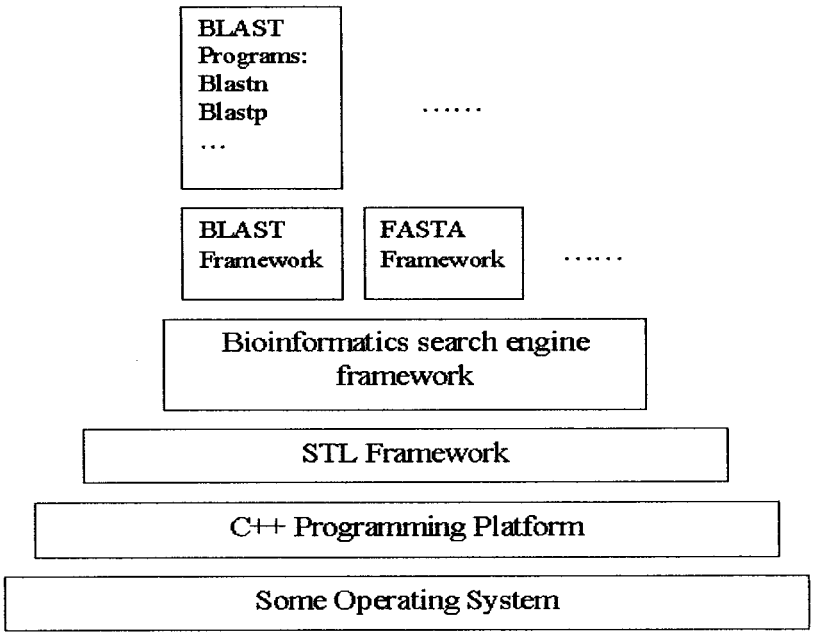

Figure 17 System Foundation and Relationship of BLAST Framework

3.4.3 First Application

According to the first step of FP (see chapter 2.2.2), I used C language to implement the first application, Blastp. This first application is regarded as the prototype that helps me to fully understand the algorithm of BLAST, and to abstract the BLAST framework. The program of Blastp completely followed the algorithm of BLAST abstracted by S. Delaney [18].

3.4.4 List of Data and Functions

According to the first application, we abstract data and functions. Here we only present the main data and functions.

Data:

Sequence is a string of STL. It is an abstraction of the query sequence and the subject sequence in the sequence database. It holds characters representing amino acids.

DBunit is a structure for the unit of the sequence database. DBunit has two attributes: sequence and its description, which are a string of STL.

DBlib is a vector of STL for the sequence database that stores DBunit.

Matrix is a template class for the score matrix, e.g. BLOSUM62. It has a two-dimension array and many member functions that can be overridden by BLAST developers.

POS_list is a vector of STL, which holds all query word positions for a neighborhood word.

FSM is a map of STL, which holds POS_lists of all neighborhood words. It represents the finite state machine.

HSPunit is a structure which records the result of hit extension, e.g. the left and right boundaries of the Maximal Segment Pair in the query sequence and subject sequence, the alignment score, etc.

HSP is a vector of STL that stores HSPunits.

Functions:

Build Neighborhood is a template function in the algorithm library. It creates the neighborhood for the query sequence depending on input data (query sequence, word size, threshold, alphabet).

Hit Detect is a template function in the algorithm Library. For each neighborhood word in FSM, it scans every subject sequence in the sequence database to find an exact match, called a Hit. Whenever it finds a Hit, it will call the function Hit Extension until all neighborhood words are scanned.

Hit Extension is a template function in the algorithm library. It is called by Hit Detect. It extends the Hit to get a Maximal Segment Pair (MSP). On completion of computing the MSP from the Hit, it will return to the function Hit Detect.

Alignment is a function defined in the Bioinformatics Search Engine framework. It executes the BLAST alignment using the input data and the chosen algorithms.

Result is a function defined in the Bioinformatics Search Engine framework. It deals with each High-scoring Segment Pair in the data container of HSP to make the report using some statistical theory.

3.4.5 The Scenario of BLAST engine

When a BLAST server receives a query task, it will load the input data, e.g. query sequence and parameter, score matrix and sequence database. Then the server will obtain an alignment from the BLAST engine by passing the input data and the

resources. The Figure 18 shows the scenario of the BLAST engine.

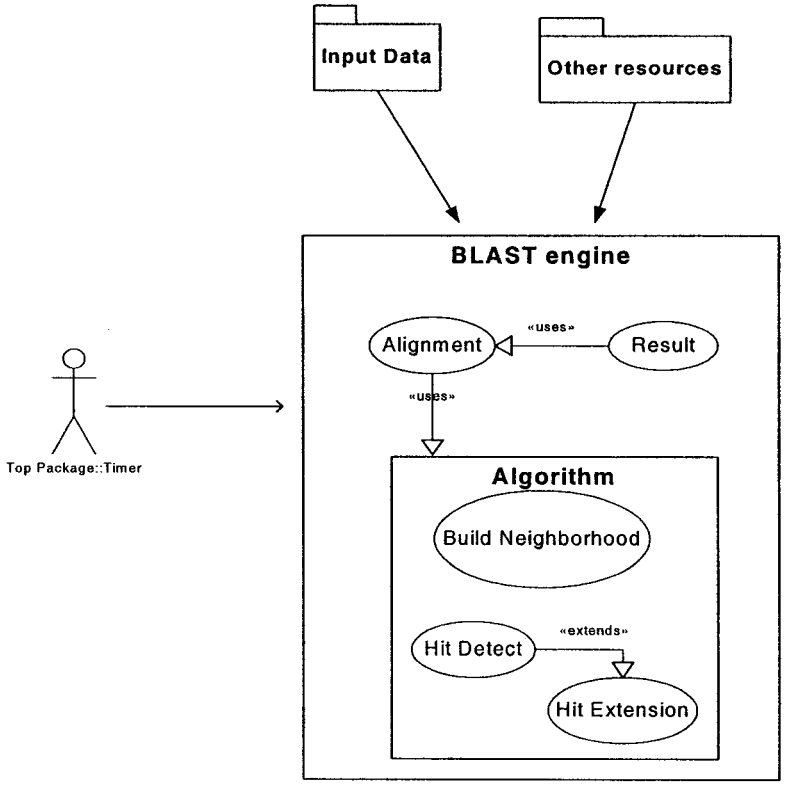

Figure 18 Use Case of BLAST Engine

1. The BLAST engine inputs all parameters into the data containers. For example, it has to load the sequence database from the database file stored in the hard disk. If the database is too large to load into the system memory at one time, the operation should be done iteratively, one sequence at a time.
2. The BLAST engine sets up an algorithm list. It chooses algorithms of Build Neighborhood, Hit Detect and Hit Extension from the algorithm library.

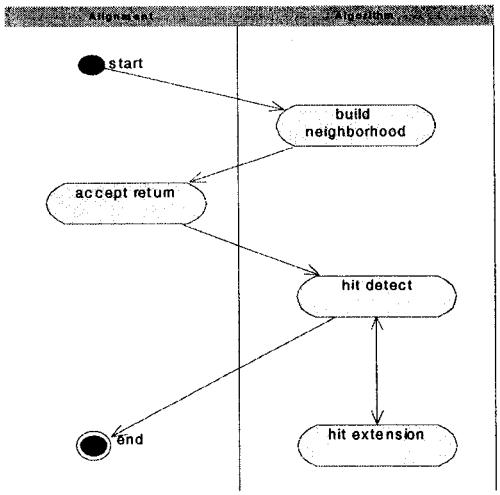

Figure 19 Activity Diagram of Alignment

3. The BLAST engine calls the function Alignment to align with the data and the chosen algorithm (see Figure 19).

   a) First, the function Alignment calls the function Build Neighborhood to make a lookup table for each query word. The function Build Neighborhood will use the given score system (the score matrix) and the neighborhood score threshold to calculate the neighborhood words for each query word. Then the neighborhood words and the related query word positions will be saved into the data container FSM.

   b) After the neighborhood construction, the function Alignment will call the function Hit Detect to scan the sequence database with the neighborhood words. For each neighborhood word in FSM, the function Hit Detect scans every subject sequence in the data container DBlib to find an exact match. Whenever it finds a Hit, it will call the function Hit Extension until all neighborhood words are scanned.

c) The function Hit Extension is called by the function Hit Detect to extend the Hit to get the Maximal Segment Pair. It extends to the left and the right in query sequence and the subject sequence until the alignment score is lower than the falloff score *X*. Whenever it gets the MSP from the Hit, it will return to the function Hit Detect.

4. The BLAST engine will call the function Result to create a report to the user. The function Result uses the complex statistical theory to deal with the data container HSP. It will calculate the Expect value and sort the result of Maximal Segment Pair by the score of Maximal Segment Pair. The output report is shown in chapter 2.1.1.

3.4.6 Pattern

From the BLAST System, we use the pattern Specification-Engine (see Figure 20). As input data, Language 1 is translated to Language 2 by the language machine. The interpreter inside the language machine works depending on the grammar. That means given a different grammar we can get a different output Language 2.

Here, the grammar becomes a template, and the interpreter does its job according to this template. A same language can be translated by the interpreter to many different languages as long as it is given enough grammars. Grammars can be components inputted from outside, which can be maintained and updated independently. Moreover, language 1 can be any kind of language, which can be interpreted to language 2 as long as given the relevant grammar.

In the Specification- Engine, the only one stable thing is the interpreter that just receives the input data- language 1 and uses a template- Grammar to output the result- language 2.

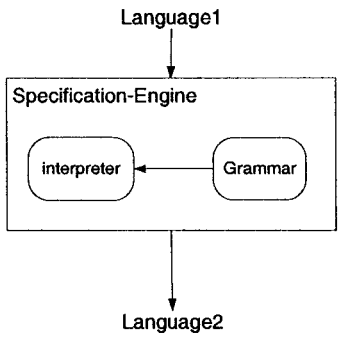

Figure 20 Pattern of Specification-Engine

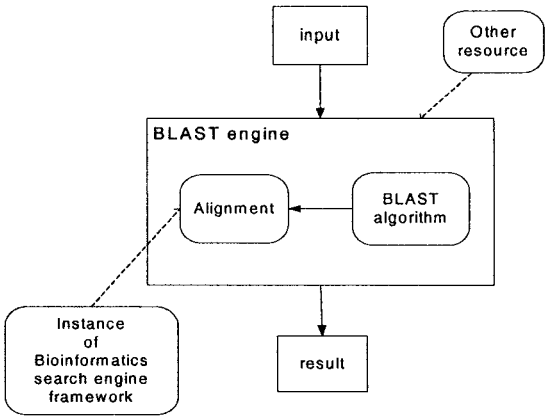

Figure 21 Possible Solution of BLAST engine

The BLAST engine is the Specification- Engine. The interpreter role is the alignment of BLAST and the grammar is the algorithm of BLAST (see Figure 21). Whatever the version of BLAST, the BLAST alignment always does the three-step job, build neighborhood, hit detect, and hit extension. If given different BLAST algorithms, the BLAST engine gives different results.

Hence, this solution satisfies the requirement of reusability. The BLAST

algorithms can be components maintained and updated independently. In this solution, the data structure and algorithm are separated which obeys the principle of Generic Programming.

3.4.7 Structure of the BLAST Framework

Figure 22 shows the simplified structure of the BLAST framework. It presents an overall structure at a high level of abstraction and hides the implementation details. This simplified structure is easy to understand and can help to form a conceptual model of the BLAST framework.

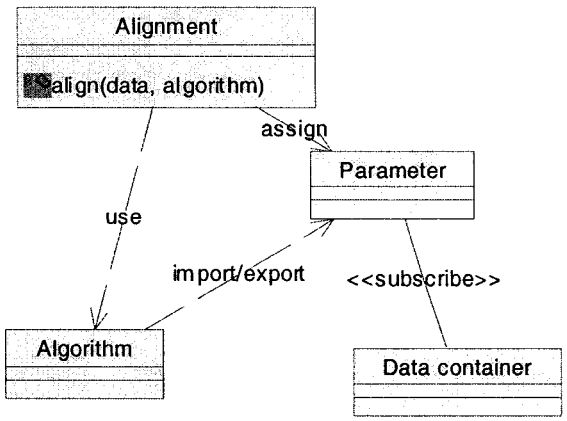

Figure 22 High Level Structure of the BLAST Framework

According to Generic Programming, the data and algorithm should be separated. There should be an agent, the Parameter class, which works as the role of iterator (see Figure 22). Algorithm does not depend on anything except the Parameter class.

The Alignment class encapsulates the control flow of BLAST engine. It executes the alignment using the given data and algorithms. The Algorithm represents the algorithm library of BLAST algorithms. For example, Build

Neighborhood, Hit Detect and Hit Extension described in 3.4.4. The Data Container defines the data structure of BLAST, including Sequence, DBlib, etc mentioned in 3.4.4. The Algorithm and the Data Container plug into the Parameter. The Data Container registers itself into the Parameter object, and the Algorithm knows nothing except the Parameter.

3.4.8 Implementation Hierarchy of the BLAST Framework

Figure 23 shows the implementation hierarchy of the BLAST framework. This implementation hierarchy shows the relationship and organization of the components of our design. The BLAST framework follows the rule of the object-oriented development that separates the definition and implementation. BLAST Programs can directly use the components, which are compiled beforehand, and also can implement the definition.

The prototype in the Figure 23 is abstracted from the real world. (In this thesis, the prototype is the first application, Blastp, implemented in C language.) We use the methodology, Framework Programming, to abstract the BLAST framework. The Bioinformatics search engine framework is only an abstract model. For the future extension, the Bioinformatics search engine framework includes two parts: interface definition and implementation. The BLAST framework includes two parts: data-container and algorithm, both of which include interface definition and implementation. As Figure 23 shows, the BLAST framework uses the components of the Bioinformatics search engine framework.

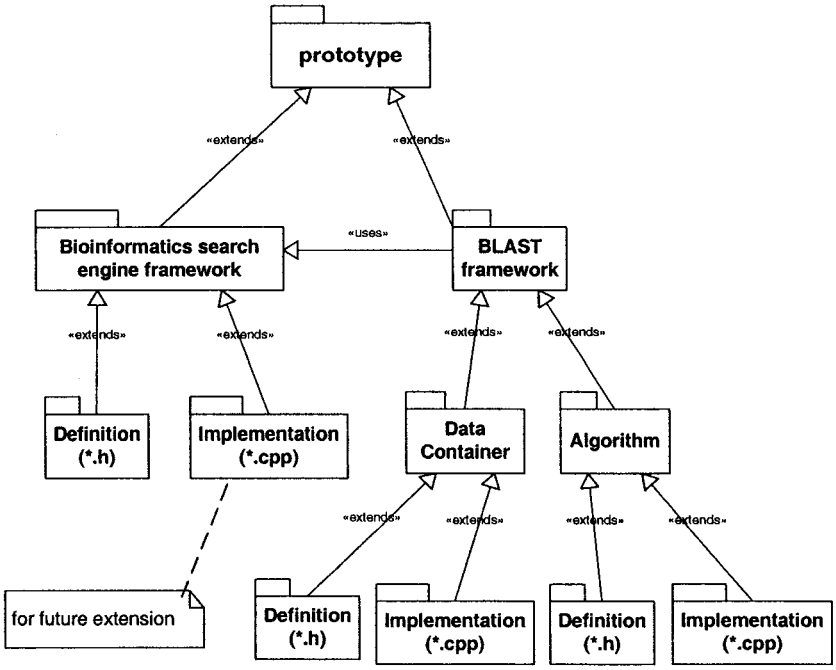

Figure 23 Implementation Hierarchy of the BLAST Framework

Chapter 4

Framework Design

This chapter gives static and dynamic models of the BLAST framework including the class model, interaction model, activity model and implementation model. Readers will see the specification of the classes as well as their relationships. This chapter is described by the static and dynamic diagrams of UML [43].

4.1 Review of the Development Process

We followed the methodology FP to design and implement the BLAST framework. First, we did the iteration for implementing and refactoring the prototype. In this process, we developed Blastp in C language. We fully understood the workflow and algorithm of BLAST, separated the hot spot, and abstracted the BLAST framework.

The next iteration was refactoring the BLAST framework. In this process, we designed the BLAST structure and classes including the Data Container and the Algorithm, and implemented Blastp again.

FP is a test-driven methodology. During the design process, we went back to the previous steps whenever the design did not pass the test.

4.2 Static Model

Figure 24 shows the overall class diagram that presents a static model of the BLAST framework. In this figure, classes of the BLAST framework are the top-level classes.

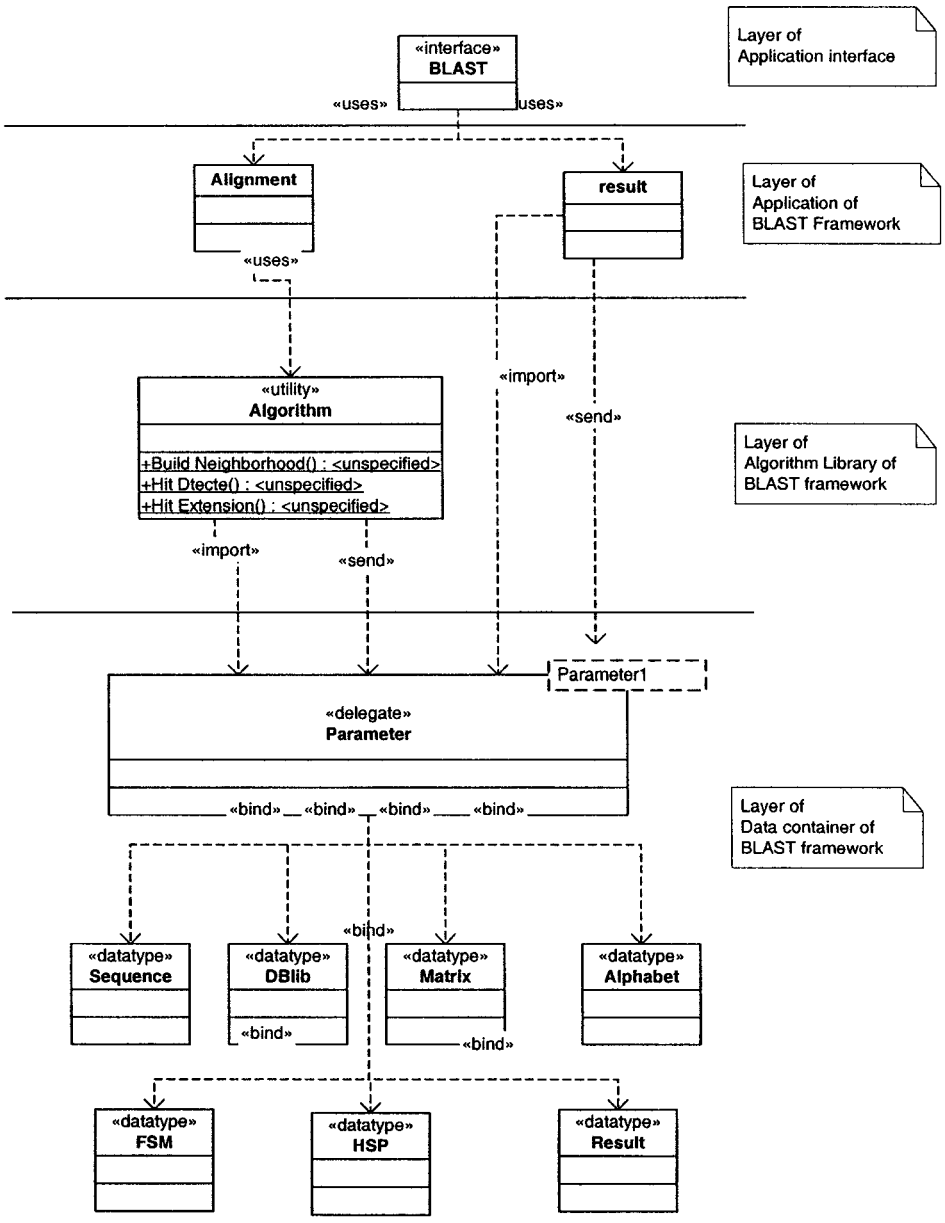

Figure 24 Overall Class Diagram

There are four packages in the overall class diagram: the Application interface, the Application of the BLAST framework, the Algorithm Library of the BLAST framework, and the Data Container of the BLAST framework.

The Application interface package includes the interface class of the BLAST

application, which depends on the Alignment class and the Result class in the package of the Application of the BLAST framework.

The package of the Application of the BLAST framework includes two classes: the Alignment class and the Result class. The Alignment encapsulates the control flow of BLAST engine, for computing the alignments using the given data and algorithms. The Result class encapsulates the statistical calculation of BLAST. It uses the result of the BLAST alignment to create a report to the user. The Result class is a subsystem of BLAST, which could be a framework.

The package of the Algorithm Library of the BLAST framework includes the Algorithm class, which defines the BLAST functions, Build Neighborhood, Hit Detect and Hit Extension described in 3.4.4. The Algorithm class and the classes of the Data Container plug into the Parameter class.

The package of the Data Container of the BLAST framework includes all the data structures of BLAST, including Sequence class, DBlib class, Matrix class, Alphabet class, FSM class, HSP class Result class, and others, which will be discussed in detail later. The most important one is the Parameter class whose role is that of be the STL iterator. The data class should register itself into the Parameter. By this means, the data structures can be connected to the Algorithm and the Algorithm depends only on the Parameter.

In the next section, we will describe the Data Container Package and the Algorithm Library package of the BLAST framework in detail.

4.3 Data Container Package of the BLAST framework

4.3.1 Structure

Figures 25 and 26 show the detailed class diagram of the Data Container Package. The Sequence class is a string of STL. The DBunit class is a structure for the unit of the sequence database. The DBlib class for the sequence database is a STL vector. The relationship of these three classes is as follows: the Sequence class is the element of the DBunit class, and the DBunit class is the element of the Dblib class.

The alphabet class is a char-type array for protein and DNA. The Matrix class is a template class. It has a two- dimensional array and many member functions that can be overridden by BLAST developers.

The Result class for storing the result of BLAST is a vector of STL. The POS_list class is a vector of STL, which holds the query word positions for a neighborhood word. The FSM class is a map of STL, which holds POS_lists of all neighborhood words. The HSPunit class is a structure which records the result of hit extension, e.g. the left and right boundaries of the Maximal Segment Pair in the query sequence and subject sequence, the alignment score, etc. The HSP class is a vector of STL that stores HSPunits.

The No_order class and the Cell class are interior classes which are only used in the process of Neighborhood Construction. The Cell class is a structure for an ordered alphabet. It is the element of the No_order class. The No_order class is a vector of STL. When building the neighborhood, the alphabet needs to be

reorganized. The No_order class stores the reorganized alphabet.

The Parameter class works as an agent of all the data class except the interior classes. For example, the DBunit class, the POS_list class, the HSPunit class, the No_order class and the Cell class.

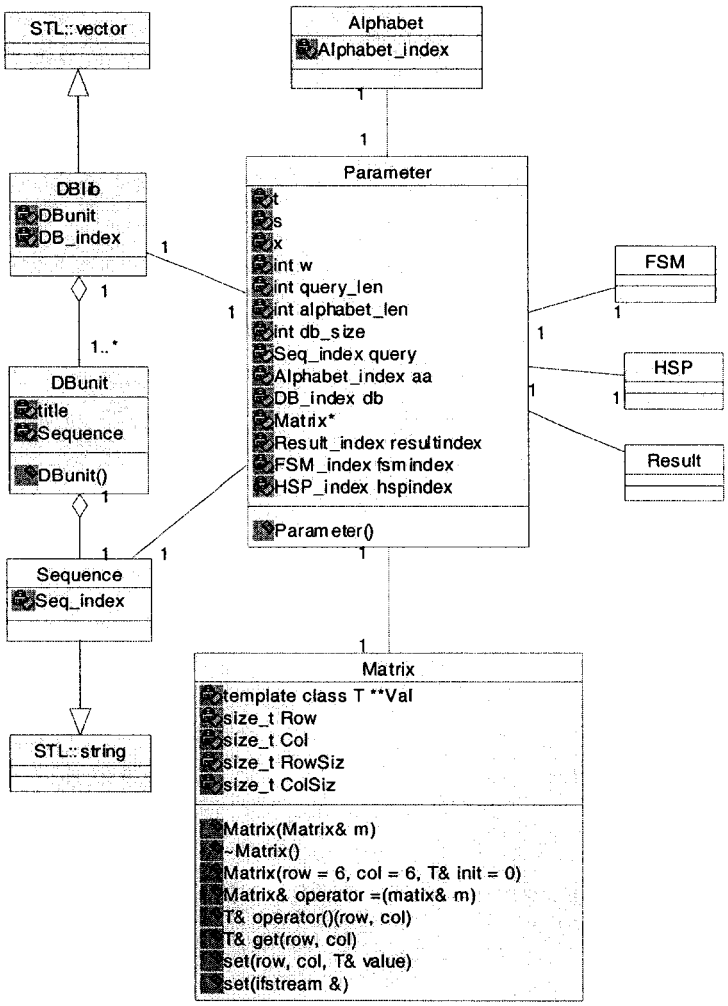

Figure 25 Class Diagram of Data Container Package - 1

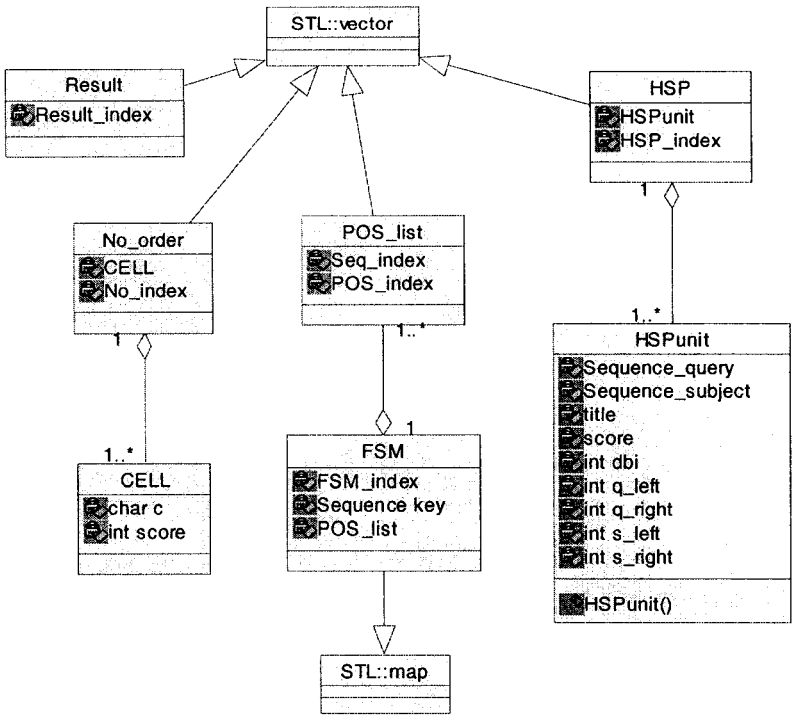

Figure 26 Class Diagram of Data Container Package - 2

4.3.2 Description of the Major Classes

4.3.2.1 The Sequence Class

Description

The Sequence class is a string of STL. It is an abstract of the query sequence and the subject sequence in the sequence database. It holds gene characters.

Code Example

```
typedef string SEQUENCE;
typedef SEQUENCE::const_iterator Seq_index;
```

4.3.2.2 The DBunit Clas

Description

The DBunit class is a structure for the unit of the sequence database. The

DBunit class has two attributes: seq and its description.

Member Data:

Title- is a STL string for the descriptions of the subject sequences in the sequence database.

Seq- is a SEQUENCE class for the subject sequence of the sequence database.

Member Function:

DBunit()- is the constructor for the DBunit class.

Code Example

```
struct DBunit{
        string     title;
        SEQUENCE   seq;

        DBunit(){title=""; seq="";};
};
```

4.3.2.3 The DBlib Class

Description

The DBlib class is a vector of STL for the sequence database that stores DBunit. The reason why we do not design a DataBase class (the code example is shown in the bad design as follows) is that we have to dynamicly assign the size of the DBlib in the program when the BLAST program runs on a personal computer. The sequence database is several hundred megabytes, e.g. nr is approximately 700MB. For a personal computer, the BLAST program cannot load the entire database into the physical memory at once. Hence, we left the function of loading the sequence database into the program.

Bad design:

```
class DataBase{
public:
    DataBase(){};
    DataBase(ifstream&);

    DBlib db;
    int dbsize;
    long DBMAXSIZE=1500000;

};
```

Code Example

```
typedef vector<DBunit> DBlib;
typedef DBlib::const_iterator DB_index;
```

4.3.2.4 The Matrix Class

Description

The Matrix class is a template class for the score system. It has a two-dimensional array and many member functions that can be overridden by BLAST developers if desired.

Member Data:

Val- is a two- dimensional array for the score matrix, e.g. BLOSUM62.

Row, Col, RowSiz, ColSiz- are the attributes of the score matrix. *Row* and *RowSiz* are the variable of the matrix row. *Col* and *ColSiz* are the variable of the matrix column.

Member Function:

matrix (const matrix<T>& m)- is a copy constructor.

matrix (size_t row = 6, size_t col = 6, const T& init=0)- is a constructor.

~matrix ()- is the destructor.

matrix<T>& operator = (const matrix<T>& m)- is the assignment operator.

T& operator () (size_t row, size_t col)- is the subscript operator.

T& get(size_t row, size_t col)- is the subscript operator.

void set(size_t row, size_t col, const T& value)- is the assignment operator.

void set(ifstream&)- is the assignment operator.

Code Example

```
template <class T>
class matrix{
private:
    T ** Val;
    size_t Row, Col, RowSiz, ColSiz;
public:
    matrix (const matrix<T>& m); // Constructors
    matrix (size_t row = 6, size_t col = 6, const T& init=0);
    ~matrix ();  // Destructor
    // Assignment operators
    matrix<T>& operator = (const matrix<T>& m);
    T& operator () (size_t row, size_t col); // Subscript operator
    T& get(size_t row, size_t col);
    void set(size_t row, size_t col, const T& value);
    void set(ifstream&);
};
```

4.3.2.5 The POS_list Class

Description

The POS_list class is a vector of STL, which holds the query word positions for a neighborhood word.

Code Example

```
typedef vector<Seq_index> POS_list;
typedef POS_list::const_iterator POS_index;
```

4.3.2.6 The FSM Class

Description

The FSM class is a map of STL, which holds the POS_lists of all neighborhood words.

Code Example

```
typedef map<SEQUENCE, POS_list> FSM;
typedef FSM::const_iterator FSM_index;
```

4.3.2.7 The HSPunit Class

Description

The HSPunit class is a structure which records result of hit extension, e.g. the left and right boundaries of the Maximal Segment Pair in the query sequence and subject sequence, the alignment score, etc.

Member Data:

Query, subject- are SEQUENCE classes for recording the query segment and the subject segment of Maximal Segment Pair (MSP).

Title- is a STL string for the description of the subject segment.

Score, dbi, q_left, q_right, s_left, s_right- are the information of each Maximal Segment Pair. *Score* is the score of MSP; *dbi* is the index of the subject segment in the sequence database; *q_left* and *q_right* are the left and right boundary of the query segment; *s_left* and *s_right* are the left and right boundary of the subject segment.

Member Function:

HSPunit()- is the default constructor.

HSPunit (const Seq_index&, const Seq_index&, const Seq_index&, const

Seq_index&, const BINT&, const string&, const int&, const int&,const int&, const int&,const int&)- is the constructor.

Code Example

```
struct HSPunit
{
    SEQUENCE query,
             subject;
    string title;
    BINT   score;
    int    dbi,
             q_left,
             q_right,
             s_left,
             s_right;
    HSPunit() {score=0;dbi=0;   q_left=0;  q_right=0; s_left=0;
             s_right=0; title=""; query=""; subject="";};
    HSPunit (const Seq_index&, const Seq_index&, const Seq_index&,
             const Seq_index&, const BINT&, const string&, const int&,
             const int&,const int&, const int&,const int& );
};
```

4.3.2.8 The HSP Class

Description

The HSP class is a vector of STL that stores HSPunits.

Code Example

```
typedef vector<HSPunit> HSP;
typedef HSP::iterator HSP_index;
```

4.3.2.9 The Parameter Class

Description

The Parameter class works as an agent for all the data class except the interior classes. For example, the DBunit class, the POS_list class, the HSPunit class, the

No_order class and the Cell class. The Algorithm class and the classes of the Data Container plug with the Parameter class. The data classes register themselves into the Parameter, and the Algorithm class only knows the Parameter.

Member Data:

T- is the neighborhood word score threshold.

S- is the alignment score threshold.

X- is the alignment falloff score.

W- is the word size.

query_len - is the length of the query sequence.

alphabet_len- is the length of the alphabet.

Db_size- is the length of the sequence database.

Queryindex- is the index of the query sequence.

Aaindex- is the index of the alphabet.

Dbindex- is the index of the sequence database.

resultindex- is the index of the result.

HSPindex- is the index of the HSP.

FSMindex- is the index of the FSM.

Member Function:

Parameter () :lq(0),la(0),dbsize(0){}-is the default constructor.

Parameter(const BINT& ,const BINT& ,const BINT& ,const BINT& ,const int & ,const int & ,const long& , const SEQUENCE& ,const SEQUENCE& ,const DBlib&)- is a constructor.

Code Example

```
struct Parameter
{
    BINT    t,  //threshold word score
            s,  //threshold alignment score
            x;  //falloff score
    int     w,  //wordsize
            query_len,
            alphabet_len;
    int     db_size;
    Seq_index  queryindex,
               aaindex;
    DB_index   dbindex;
    Result_index resultindex;
    HSP_index  hspindex;
    FSM_index  fsmindex;
    Parameter () : query_len (0), alphabet_len (0),db_size(0){};
    Parameter(const  BINT&  ,const  BINT&  ,const  BINT&  ,const
          BINT& ,const int & ,const int & ,const long& , const
          SEQUENCE& ,const SEQUENCE& ,const DBlib& );
};
```

4.4 Algorithm Library package of the BLAST framework

4.4.1 Structure

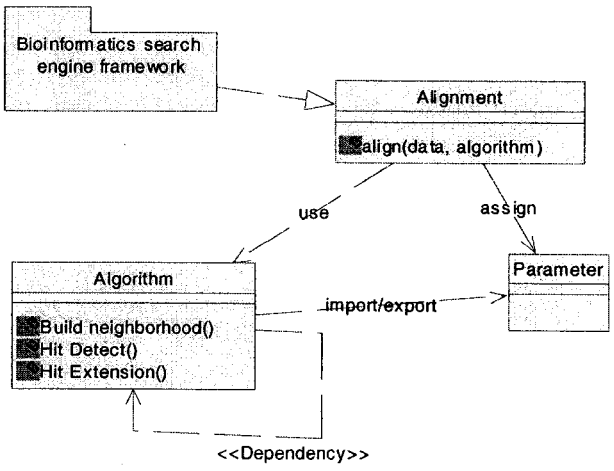

Figure 27 Class Diagram of the Algorithm Class

Figure 27 shows the class diagram of the Algorithm Library package of the BLAST framework. The Alignment class is derived from the Bioinformatics search engine framework. The Alignment class assigns to the Parameter class the input data and uses the Algorithm class for doing an alignment. The Algorithm class retrieves and places data into the Data Container through the Parameter class. The Algorithm class depends on itself since some algorithms need to use other algorithms in the Algorithm Library. For example, the function Hit Detect invokes the function Hit Extension. The table 4 shows the function list of the Algorithm class.

Table 4 Function List of the Algorithm Class

Class Name	Algorithm
Responsibility	BLAST algorithm library
Attributes	
Operation	
Build neighborhood()	To build neighborhood (see 4.4.2)
Hit Detect()	To Hit Detect (see 4.4.2).
Hit Extension()	To Hit Extension (see 4.4.2).
Relationship	

4.4.2 Description of the Algorithm Class

The Algorithm class defines the BLAST algorithms, for example, Build Neighborhood, Hit Detect and Hit Extension described in 3.4.4. The Algorithm class and the classes of the Data Container plug together with the Parameter class.

4.4.2.1 The Function Build Neighborhood

Description

Build Neighborhood is a template function in the algorithm library. It creates the

neighborhood for the query sequence depending on input data (query sequence, word size, threshold, alphabet). The query sequence Q is scanned for all words V of length W, and computes each neighbor word U of V that scores at least T. Each word V may have zero or more neighbors. The set of neighbors of all query words is the neighborhood N.

Code Example

```
template <class T1, class T2>
void buildNei(T1& , T2&);
```

The class T1 is the input data, and the class T2 is the output data.

4.4.2.2 The Function Hit Detect

Description

Hit Detect is a template function in the algorithm Library. Using the FSM of the neighborhood, it scans every subject sequence in the sequence database to find an exact match, called a Hit. Whenever it finds a Hit, it will call the function Hit Extension.

Code Example

```
template <class T1, class T2 >
void hit(T1& , T2&);
```

The class T1 is the input data, and the class T2 is the output data.

4.4.2.3 The Function Hit Extension

Description

Hit Extension is a template function in the algorithm library, called by Hit Detect. It extends the Hit to get a Maximal Segment Pair (MSP). It extends to the

left and the right in the query sequence and the subject sequence until the alignment score is lower than the falloff score *X*. Once it completes the MSP from the Hit, it will return to the function Hit Detect.

Code Example

```
template <class T1, class T2 >
void extend(T1& , T2& );
```

The class T1 is the input data, and the class T2 is the output data.

4.5 Dynamic Model

Figure 28 shows how several objects of BLAST collaborate in a single use case. It consists of a set of BLAST objects and their relationships, and the messages that may be dispatched among them. Figure 28 is a sequence diagram that emphasizes the time ordering of messages. It is a dynamic model of BLAST alignment.

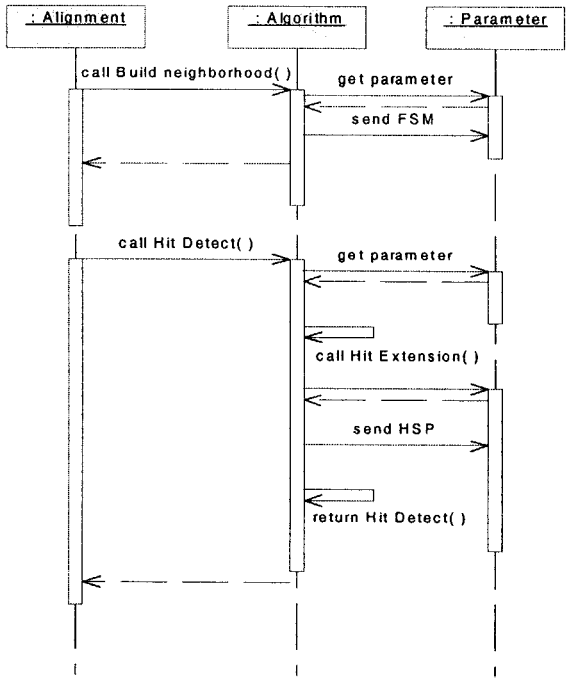

Figure 28 Sequence Diagram of BLAST Alignment

The scenario of BLAST alignment:

1. After given an input assignment, the Alignment class will call the function Build Neighborhood in the Algorithm class.
2. The function Build Neighborhood responds the call and it will get the data through the Parameter class.
3. When the function Build Neighborhood has finished the neighborhood table, it will save the neighborhood table into the data container FSM through the Parameter class.
4. After the function Build Neighborhood finishes its work completely, it will return to the Alignment class.
5. When the Alignment class accepts the return of the function Build Neighborhood, it will call the function Hit Detect in the Algorithm class to scan the subject sequence database for each neighborhood word.
6. The function Hit Detect responds the call and it will get the data through the Parameter class.
7. The function Hit Detect does the scan task. When it finds a Hit, an exact match for a neighborhood word, it will call the function Hit Extension in the Algorithm class to extend the Hit to get a Maximal Segment Pair.
8. The function Hit Extension responds the call and it will get the data through the Parameter class.
9. When the function Hit Extension has done the extension, it will save the Maximal Segment Pair (MSP) into the data container HSP through the

Parameter class. If it does not find the MSP (if the alignment score is lower than the alignment score threshold *S*, the segment pair is not MSP), it will return to the function Hit Detect.

10. After the function Hit Extension finishes its extension, it will return to the function Hit Detect.
11. Go to the step 7 until all the neighborhood words are scanned.
12. After the function Hit Detect finishes its scan work, it will return to the Alignment class.
13. A BLAST alignment will finish. The process goes to the statistical analysis for the result report.

4.6 Implementation Model

Figures 29, 30 show a component diagram and a deployment diagram, found in modeling the physical aspects of object-oriented systems. Figure 29 shows the organization and dependencies among a set of components in the BLAST framework. Figure 30 shows the configuration of run time processing nodes for a BLAST system and the BLAST components that live on them. This section uses Blastp as an example to specification.

4.6.1 Component Diagram

According to the implementation hierarchy of the BLAST framework (see chapter 3.4.8), the BLAST framework separates the definition components and

implementation components (see Figure 29).

The BLAST framework derives from the Bioinformatics search engine framework. The BLAST framework has two components: the BLAST Algorithm and the BLAST Data Container. Each one has the definition and implementation components. Their Definition components derive from the Bioinformatics search engine framework. The implementation component of the BLAST Algorithm stores all components of BLAST algorithms. The implementation component of the BLAST Data Container stores all the implementation of the BLAST Data Container member functions. Developers can add or maintain the components of the BLAST framework easily. Table 5 gives the list of components of the BLAST framework.

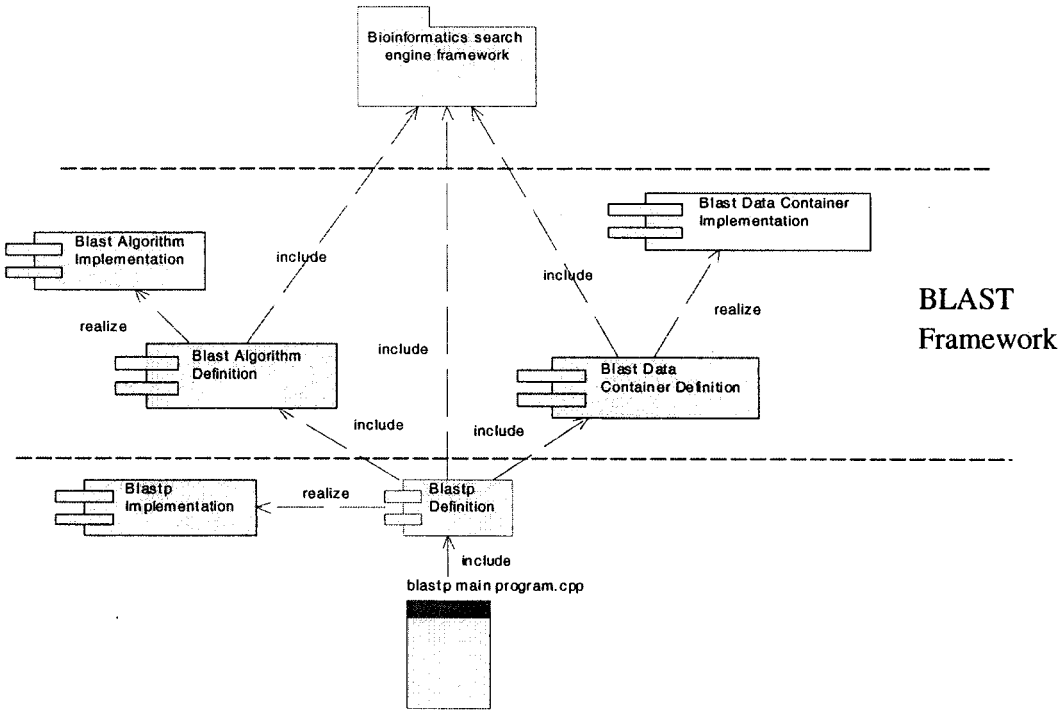

Figure 29 Component Diagram of Blastp

Table 5 List of Components of the BLAST framework

Component	Included Classes	Contain Classes	Included File
BLAST Data Container Definition	Parameter, Data container	See chapter 4.3	Bioinformatics Search Engine Framework. h
BLAST Algorithm Definition	Algorithm	See chapter 4.4	Bioinformatics Search Engine Framework. h
BLAST Data Container Implementation		Function realization	BLAST Data Container. h
BLAST Algorithm Implementation		Function realization	BLAST Algorithm. h

To implement the Blastp program in BLAST server (see Figure 29), the first thing is to include the definition into the header file of Blastp (blastp.h) from the BLAST framework. The second thing is to customize the function of the class derived from the BLAST framework in the implementation file of Blastp (blastp.cpp). the final step is to program the main program of Blastp.

4.6.2 Deployment Diagram

In the real world, the BLAST server can provide many BLAST services, e.g. Blastp. The system hardware of the BLAST server could be a mainframe, a PC or a set of servers. The BLAST system that runs on the distributed network platform is composed by a set of distributed servers. Each server has its special duty. Figure 30 shows the distributed system of BLAST, using the service of Blastp as an example.

Table 6 List of Blastp Deployment

Node	BLAST Server
Executable Files	Blastp .exe

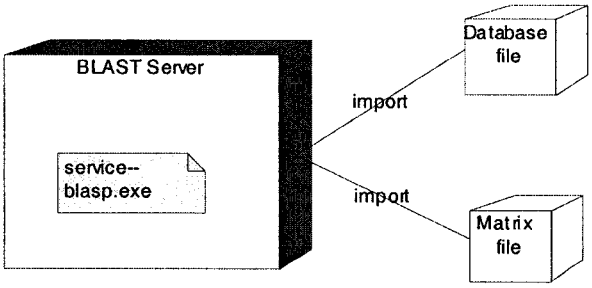

Figure 30 Deployment Diagram of Blastp

The duty of the BLAST server is to provide the service of BLAST, e.g. Blastp. The duty of other servers is to provide resources for the BLAST server. In Figure 30, the BLAST server for doing the Blastp needs a server to provide the Database file and another server to provide the Matrix file. These servers collaborate to implement the function of BLAST.

4.7 System Programming Environment

Figure 31 shows the organization of programming environment, which follows the principle of CVS[7]. There are four parts in the programming environment of BLAST: *doc* , *internet, src* and *test.* In *test*, there are also four parts: *bin, build, config* and *src*. The folder *doc* of BLAST stores the document of developing and debugging, and some articles of BLAST and guide manual. The folder *Internet* holds the resource from Internet. The folder *src* saves the source code of BLAST from other open sources.

The folder *bin* of *test* is to store executable file of BLAST components. The folder *build* has a few versions of the *makefile*. The folder *config* is for the files *profile* and *config*. The folder *src* of *test* saves every version of source code of BLAST ordered by date. In every folder of them, there are folders *inc* and *lib* which include files and library files.

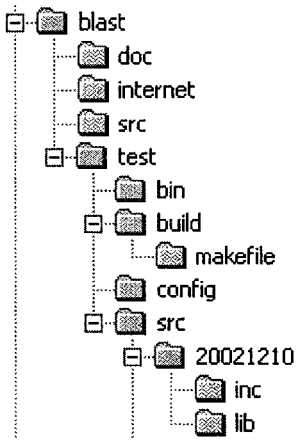

Figure 31 Organization of Programming Environment

Chapter 5

Cookbook and Example

This chapter gives recipes and examples to show how to use and extend the BLAST framework. The code examples in this chapter are C++ language. The definition and implementation in the chapter are on the implementation model of the BLAST framework (see chapter 4.6).

5.1 Framework Cookbook

This section provides a guideline on how to customize the BLAST framework. It introduces a set of recipes, each of which is description of a typical customization on one aspect.

5.1.1 Recipe 1: Overview of the Cookbook

This first recipe gives an overview of this cookbook. This cookbook discusses two recipes: customize the BLAST Algorithm component and customize the BLAST Data Container component. Each recipe follows a template including its purpose, steps involved in customization, and cross-reference to other recipes.

5.1.2 Recipe 2: Customize the BLAST Algorithm Component

Purpose Define and implement the three methods of the BLAST Algorithm: Build Neighborhood, Hit Detect, Hit Extension. These methods can be overridden.

Steps

1. Define Build Neighborhood in the component of the BLAST Algorithm Definition.

```
template <class T1, class T2>
void buildNei(T1& , T2&);
```

   Implement it in the component of the BLAST Algorithm Implementation.

2. Define Hit Detect in the component of the BLAST Algorithm Definition

```
template <class T1, class T2>
void hit(T1& , T2&);
```

   Implement it in the component of the BLAST Algorithm Implementation.

3. Define Hit Extension in the component of the BLAST Algorithm Definition

```
template <class T1, class T2>
void extend(T1& , T2&);
```

   Implement it in the component of the BLAST Algorithm Implementation.

Cross-Reference None

5.1.3 Recipe 3: Customize the BLAST Data Container Component

Purpose Define the classes in BLAST Data Container component: HSPunit, DBunit, Parameter and Matrix.

Steps

1. Define HSPunit in the component of the BLAST Data Container Definition

   a) Define Member Data:

      Query, subject- are SEQUENCE classes for recording the query segment and the subject segment of Maximal Segment Pair (MSP).

      Title- is a STL string for the description of the subject segment.

Score, dbi, q_left, q_right, s_left, s_right- are the information of each Maximal Segment Pair.

b) Define Member Function:

HSPunit()- is the default constructor.

HSPunit (const Seq_index&, const Seq_index&, const Seq_index&, const Seq_index&, const BINT&, const string&, const int&, const int&,const int&, const int&,const int&)- is the constructor.

c) Implement member functions in the component of the BLAST Data Container Implementation.

2. Define DBunit in the component of the BLAST Data Container Definition

a) Define Member Data:

Title- is a STL string for the descriptions of the subject sequences in the sequence database.

Seq- is a SEQUENCE class for the subject sequence of the sequence database.

b) Define Member Function:

DBunit()- is the constructor for the DBunit class.

c) Implement member functions in the component of the BLAST Data Container Implementation.

3. Define Parameter in the component of the BLAST Data Container Definition

a) Define Member Data:

T- is the neighborhood word score threshold.

S- is the alignment score threshold.

X- is the alignment falloff score.

W- is the word size.

query_len - is the length of the query sequence.

alphabet_len- is the length of the alphabet.

Db_size- is the length of the sequence database.

Queryindex- is the index of the query sequence.

Aaindex- is the index of the alphabet.

Dbindex- is the index of the sequence database.

resultindex- is the index of the result.

HSPindex- is the index of the HSP.

FSMindex- is the index of the FSM.

b) Define Member Function:

Parameter () -is the default constructor.

Parameter(const BINT&, const BINT&, const BINT&, const BINT&, const int&, const int&, const long&, const SEQUENCE&, const SEQUENCE&, const DBlib&)- is a constructor.

c) And implement member functions in the component of the BLAST Data Container Implementation.

4. Define Matrix in the component of the BLAST Data Container Definition

a) Define Member Data:

Val- is a two- dimensional array for the score matrix.

Row, Col, RowSiz, ColSiz- are the attributes of the score matrix.

b) Define Member Function:

matrix (size_t row = 6, size_t col = 6, const T& init=0)- is a constructor.

matrix (const matrix<T>& m)- is a copy constructor.

~matrix ()- is the destructor.

matrix<T>& operator = (const matrix<T>& m)- is the assignment operator.

T& operator () (size_t row, size_t col)- is the subscript operator.

T& get(size_t row, size_t col)- is the subscript operator.

void set(size_t row, size_t col, const T& value)- is the assignment operator.

void set(ifstream&)- is the assignment operator.

c) Implement member functions in the component of the BLAST Data Container Implementation.

Cross-Reference None

5.2 Example of Framework Modification

This section will illustrate how to build and modify a BLAST program using the BLAST framework. All illustrations are based on the previous recipes for customizing individual components. We will not dive into details except when necessary.

Suppose we want to develop a program of Blastp. Several steps are sketched out as follows:

Step 1: Customize the BLAST Algorithm Component

1. Define and implement Build Neighborhood in the component BLAST Algorithm.
2. Define and implement Hit Detect in the component BLAST Algorithm.
3. Define and implement Hit Extension in the component BLAST Algorithm.

Step 2: Customize the BLAST Data Container Component

1. Define and implement Matrix in the component BLAST Data Container.
2. Define and implement Parameter in the component BLAST Data Container.
3. Define and implement HSPunit in the component BLAST Data Container.
4. Define and implement DBunit in the component BLAST Data Container.
5. Define SEQUENCE in the component BLAST Data Container Definition.
6. Define DBlib in the component BLAST Data Container Definition.
7. Define Alphabet in the component BLAST Data Container Definition.
8. Define HSP in the component BLAST Data Container Definition.
9. Define POS_list in the component BLAST Data Container Definition.
10. Define FSM in the component BLAST Data Container Definition.
11. Define CELL in the component BLAST Data Container Definition.
12. Define No_order in the component BLAST Data Container Definition.

Step 3: Load the Input Data into the Data Container

This step instantiates the objects from Data classes, and initializes, and assigns data.

1. Instantiate and initialize the Matrix class.

```
typedef matrix <int> Mymatrix;
Mymatrix Matrix(MatrixSize, MatrixSize, -50);
```

Then assign it data from the matrix file.

```
//fmatrix is a pointer of matrix file.
Mymatrix.set(ifstream& fmatrix);
```

2. Instantiate, initialize and assigne the SEQUENCE vector for the query sequence.

```
SEQUENCE q="";
fquery>> q;   // fquery is a pointer of query file.
```

3. Instantiate, initialize and assigne the Parameter class.

```
Parameter para(atoi(argv[4]), atoi(argv[5]), atoi(argv[6]),
atoi(argv[7]), q.size(), a.size(), 0, q, a, db);
```

4. Instantiate and initialize the DBlib vector for the sequence database.

```
DBlib db;
```

5. Instantiate and initialize the FSM map for the neighborhood.

```
FSM fsm;
```

6. Instantiate and initialize the HSP vector for the HSP.

```
HSP hsp;
```

Step 4: Customize the Alignment Class

There is only one method in the Alignment class. The definition as follow:

```
template <class T1, class T2>
align(T1&, T2&);
```

1. Implement align(T1&, T2&). In order to do that, the algorithms have to be available.

```
template <class T1, class T2>
align(T1& data, T2& algorithm ){
…
case 0: buildNei(data);
        hit(data);
        break;
…
default:
…
};
```

2. Instantiate the Alignment class.

```
Alignment doBLAST;
```

Step 5: Align with Data and Algorithm.

1. Assign the parameter:

```
int algorithm=0;
```

2. Align with data and algorithm.

```
doBLAST.align(para, algorithm);
```

Step 6: Deal With the Result.

1. Customize the Result class. There is only one method in the Result class-outputData (data). The definition as follow:

```
template <class T1, class T2>
void outputData(T1&, T2&);
```

Implement this method.

2. Instantiate the Result class.

```
Result doResult;
```

3. Deal with the result.

```
doResult.outputData(para);
```

5.2.1 More Modify Examples

Example A: Add a New Algorithm

To add a new component of algorithm into the BLAST framework one needs to extend the algorithm interface definition, and implement it into the package of the BLAST Algorithm Implementation.

Code Example:

```
template <class T1, …>
yourNewAlgorithm( T1&,…);
```

Example B: Add a New Data Structure

To add a new data structure into the BLAST framework one needs to extend the component BLAST Data Container Definition, and implement the methods into the package BLAST Data Container Implementation. Then a new related attribute needs to add into the Parameter class.

Code Example:

```
template <class T1, …>
class yourNewData{ };
```

Chapter 6

Conclusion

In this thesis, we designed and implemented a BLAST framework to meet the goal of reusability of BLAST. We first built a prototype, Blastp, using the C language. From this prototype, we abstracted the BLAST framework. In the future, a set of applications should be built using the BLAST framework in order to evaluate this framework.

6.1 Summary of the work

Faster algorithms are always required because sequence databases grow dramatically. However, today's BLAST programs cannot be quickly adapted. The primary problem is that most of BLAST programs are implemented using Procedural Programming and lack reusability. NCBI (National Center for Biotechnology Information) has become aware of this problem, and begun to refactor the BLAST system. They were going to, firstly refactor the BLAST programs by object-oriented technology and secondly to go to open source.

This thesis proposes a reusable architecture for refactoring the BLAST systems. Our works make three contributions to the fields of bioinformatics and software engineering: (1) an object- oriented BLAST framework, (2) a case study implementation of the Blastp program and (3) a framework development methodology- Framework Programming.

6.1.1 Summary of Framework Programming

In the project of Know-It-All (KIA), Dr. Butler and his phd students did the research for the framework development and evolution. The Framework Programming is based on their research [22] and also on Kent's methodology of eXtreme Programming [19]. Framework Programming is a three-step process:

Step 1: iteration
- 1.1 iteration
  - 1.1.1 build or refactor the prototype
  - 1.1.2 test
- 1.2 iteration
  - 1.2.1 abstract or refactor hooks and hot spots
  - 1.2.2 test
- 1.3 test, go back 1.1 or 1.2.

The prototype could be a use case system or an executable system. To developing a framework, design patterns and Generic Programming are used to providing good solutions. Moreover, its development team model is Triangle Programming. This model gives the customer more rights to participate in development.

6.1.2 Summary of the BLAST Framework

We followed the methodology of FP to develop the BLAST framework. Firstly, we used the C language to program the Blastp as a prototype. It helped us to understand the algorithm of BLAST. Secondly, we abstracted the hooks and hot spots. Then we did the iteration of refactoring the framework.

The BLAST framework uses the principle of Generic Programming to separate data structures and algorithms into two packages: the BLAST Data Container and the BLAST Algorithm. The BLAST Data Container and the BLAST Algorithm do

not know each other and they can plug together using the Parameter class that works as the iterator of Generic Programming.

We believe this framework promotes the reusability of the system and is good for clarity. The BLAST framework is easy to use and it makes the maintenance work of BLAST easier. To evaluate the BLAST framework, we used it to build a standard program of Blastp.

6.1.3 Summary of Programming Blastp

We programmed a standard Blastp using C++ and STL [9,10] based on the BLAST framework. The algorithms of Blastp came from Delaney's research [18]. Firstly, we customized the BLAST Algorithm component. Secondly, we customized the BLAST Data Container component. Finally, we made the main program of Blastp (see chapter 5.2.1). The development of Blastp is a case study implementation that shows the BLAST framework is easy to use.

We understand the importance of efficiency for the BLAST system. However, the duty of the thesis is to provide solution of the BLAST framework for the system reusability. The Blastp program we made is to show a case study and to evaluate the BLAST framework. We did not do the optimization of the performance of the BLAST system. In this thesis, we did the test our Blastp against on the NCBI Blastp v2.2.4. That just shows readers a general picture of efficiency of our BLAST system.

We used a 348-letter sequence as the query sequence, the score matrix of BLOSUM62 as the scoring system, and the *nr* database in GenBank as the sequence

database. The program run on the Sun-Fire-280R computer (2x UltraSPARC-III+ CPU, 4G memory). All parameters were default. The *nr* database is the non-redundant GenBank CDS which is composed by translations, PDB, SwissProt, PIR and PRF. It has 1,472,604 sequences, which are 474,795,797 total letters. The number of HSP's successfully found is 407, while the number for our Blastp is 981. We use the time command to calculate the execution time. The NCBI Blastp spent 6.9×10^4 ms on the search, while our Blastp spent 2.5×10^5 ms. The execution time of our Blastp is more than three times that of the NCBI Blastp.

The algorithms of the two Blastp programs are different. Our Blastp uses the standard algorithms of NCBI Blastp v 1.4, which is a non-gapped and single hit algorithm. The algorithm of NCBI Blastp v 2.2.4 is a gapped and two- hit algorithm. A multiple hit algorithm is an optimized algorithm that speeds up by pruning the lower length MSP (Maximal Segment Pair). In general, NCBI Blastp v 2.2.4 is three times faster than NCBI v 1.4 implementation.

If our Blastp uses the optimized algorithms in the future, it will get better performance.

6.2 Lessons Learned about Software Development

The BLAST framework welcomes 'change'. In the whole project, we followed the FP methodology, and iterated nine times. The framework and programs grew up from prototype to real code. First, the first application was built, the role of which is a specific example to understand the requirement and workflow of BLAST. Then,

the first prototype of use case system of BLAST framework was abstracted from the first application. This was a confused time that lasted almost three months. Most of time was spent on debugging. The reason why it took such a long time is, I analyze here, that the real "true" feedback could not be received. The work model of FP (see 2.2.2.3) is triangle programming, which needs a three-developer team to work together. However, because this project is an individual master thesis, I had to act as both the customer and pair programmers. I could not get real feedback, and in fact, it was not the development of Triangle model, but was the activity of one programmer debugging.

The second to the ninth iteration was a happy time. Actually, I almost forgot the previous agony and enjoyed this programming. The time of each iteration was from one to seven days. The reason why it was so fast (it should be one or two weeks) is that, I thought, it was the same reason as presented above. The feedback of one programmer is three times faster than the one of three developers. This software development helps me to understand the importance of feedback, and also to notice the importance of methodology.

6.3 Future Work

This is only the first step of the development of the BLAST framework because the methodology and technology of framework or reusability are not mature. That means there exists lots of opportunity of extension.

The future work lies in two categories: refactoring of FP and refactoring of the

BLAST framework. The application requires updating the methodology. In the future, a set of applications should be built using the BLAST framework in order to evaluate this framework. The more applications are put into the work, the more mature will be the framework and the methodology.

Bibliography

[1] S. Altschul, W. Gish, W. Miller, E. Myers and D. Lipman, ***Basic local alignment search tool.*** Journal of Molecular Biology, 215(3) 403-410. Oct 1990

[2] Description of BLAST Services. URL http://www.ncbi.nlm.nih.gov/BLAST/. (Also see the NCBI C Toolkit Source Browser at http://www.ncbi.nlm.nih.gov/IEB/ToolBox/SB/hbr.html)

[3] Documentation and manuscripts of WU-BLAST. URL http://blast.wustl.edu (also see source code at http://blast.wustl.edu/blast-1.4/)

[4] RD Bjornson, AH Sherman, SB Weston, N. Willard, J. Wing. ***TurboBLAST: A Parallel Implementation of BLAST Built on the TurboHub***. TurboGenomics, Inc. http://www.hicomb.org/HiCOMB2002/papers/HICOMB2002-01.pdf. (Also see TurboBLAST web site. URL http://www.turboworx.com/solutions/turboblast)

[5] Aaron E. Darling, Lucas Carey, Wu-chun Feng. ***The design, implementation, and evaluation of mpiBLAST.*** The ClusterWorld 2003 conference. URL http://mpiblast.lanl.gov/background.html

[6] Hinkmond Wong. ***Developing Jini applications using J2ME technology.*** Boston, MA : Addison-Wesley, c2002

[7] Karl Franz Fogel, ***Open Source Development with CVS: Learn How to Work With Open Source Software***. The Coriolis Group.1999.
(Also see the document of CVS at http://www.cvshome.org/docs/ and the document in URL http://www.loria.fr/~molli/cvs/doc/cvs_toc.html)

[8] Gregory F. Rogers. ***Framework-based software development in C++ .*** Upper Saddle River, N.J. : Prentice Hall PTR, c1997

[9] Matthew H. Austern. ***Generic Programming and the STL: Using and Extending the C++ Standard Template Library*** Reading, MA.: Addison-Wesley, c1999

[10] David R. Musser, Gillmer J. Derge, Atul Saini. ***STL tutorial and reference guide : C++ programming with the standard template library ; [foreword by Alexander Stepanov]*** Boston : Addison-Wesley, c2001. (Also see Musser's page http://www.cs.rpi.edu/~musser/gp/)

[11] D. Benson, M. Boguski, D. Lipman, J. Ostell and B. Ouellette. ***GenBank.***

Nucleic Acids Research, 26(1): 1-7, 1998.

[12] G. Stoesser, W. Baker, A. Broek, E. Camon, M. Garcia-Pastor, et al., ***The EMBL nucleotide sequence database***. Nucleic Acids Research, 29(2000): 17-21.

[13] Y. Tateno, S. Miyazaki, M. Ota, H. Sugawara, and T. Gojobori, ***DNA Data Bank of Japan (DDBJ) in collaboration with mass sequencing teams.*** Nucleic Acids Research, 28(2000): 24-26.

[14] BLAST Frequently Asked Questions (FAQ) of NCBI, in URL http://www.ncbi.nlm.nih.gov/BLAST/blast_FAQs.html

[15] S. B. Needleman and C. D. Wunsch, ***A general method applicable to the search for similarities in the amino acid sequences of two proteins.*** Journal of Molecular Biology, 48: 443-453. 1970.

[16] T. F. Smith and M. S. Waterman, ***Identication of common molecular subsequences.*** Journal of Molecular Biology,147:195-197. 1981.

[17] W. R. Pearson and D. J. Lipman, ***Improved tools for Biological Sequence Comparison.*** Proc. Nucleic Acids Research. USA, 85: 2444-2448. 1988.

[18] S. Delaney, G. Butler, C. Lam, L. Thiel. ***Three improvements to the BLASTP Search of Genome Databases.*** **Proceedings of the 12th International Conference on Scientific and Statistical Database Management, (Berlin, July 26-28, 2000),** Oliver Gunther and Hans-J. Lenz (eds), IEEE Computer Society, Los Alamitos, CA, 2000, PP. 14-24.

[19] Kent Beck, ***Extreme Programming Explained: Embracing Change.*** Addison-Wesley, 2000. (Also see eXtreme Programming (XP) in URL http://www.extremeprogramming.org/)

[20] The documentation of UML of OMT in URL http://www.omg.org/technology/documents/modeling_spec_catalog.htm

[21] G. Butler, L. Li, I.A. Tjandra: ***Reusable Object-Oriented Design***. Centre Interuniversitaire en Calcul Mathématique Algébrique and Department Of Computer Science of Concordia University. 1995

[22] G. Butler, P. Dénommée. ***Documenting Framework to Assist Application Developers, Chapter of Building Application Frameworks: Object-Oriented Foundations of Framework Design, edited by Mohamed E. Fayad, Douglas C. Schmidt, Ralph E. Johdson.*** John Wiley & Sons. Inc., 1999.

[23] F. Bushmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal: ***Pattern-Oriented Software Architecture: A System of Patterns***. John Wiley & Sons. Inc., 1996.

[24] M. Fowler: ***The New Methodology***. June 2002. URL http://www.martinfowler.com/articles/newMethodology.html

[25] Greg Butler, Ling Chen, Xuede Chen, Ashraf Gaffar, Jinmiao Li, Lugang Xu, ***The Know-It-All Project: A Case Study in Framework Development and Evolution/ Domain Oriented Systems Development: Perspectives and Practices,*** *Kiyoshi Itoh, Satoshi Kumagai, T. Hirota (eds),* Taylor and Francis Publishers, UK, 2002.

[26] OMG: ***Introduction to OMG's Unified Modeling Language. 2002.*** URL http://www.omg.org/gettingstarted/what_is_uml.htm

[27] OMG: ***UML summary(v1.4). Sep.2001***. URL http://www.omg.org/cgi-bin/doc?formal/01-09-72

[28] Sinan Si Alhir: ***Understanding the Unified Modeling Language (UML). April 1999.*** Published in "Methods & Tools". URL http://home.earthlink.net/~salhir/UnderstandingTheUML.PDF

[29] Definition of Jini Technology Core Platform:"Technology Specifications" URL http://wwws.sun.com/software/jini/jini_technology.html

[30] The white paper of Jini network technology. URL http://wwws.sun.com/software/jini/whitepapers/index.html

[31] The white paper of Universal Plug and Play(UPnP). Microsoft. June, 2000. http://www.upnp.org/download/UPNP_UnderstandingUPNP.doc
(Also see the standards and documents of UPnP at http://www.upnp.org/resources/)

[32] Bob Pascoe, ***Salutation-Lite :Find-And-Bind™ Technologies For Mobile Devices,***A Salutation White Paper. June 6, 1999. URL http://www.salutation.org/lite/litesource.htm. (Also see Salutation Technology Quick Study at http://www.salutation.org/techno.htm)

[33] Christopher Renner, ***An introduction to SLP***, 2000. http://www.lkn.e-technik.tu-muenchen.de/~chris/slp/IntroSLP.html. (Also see web resource in URL http://www.srvloc.org/)

[34] Bluetooth V1.1 Specification (Core) part E: Bluetooth SDP. URL https://www.bluetooth.org/foundry/specification/document/specification. (Also see

the document in URL http://studwww.ira.uka.de/~s_oden/18c3/html/slide_45.html)

[35] S. D. Gribble and et al. ***The ninja architecture for robust internet-scale systems and services.*** Special Issue of IEEE Computer Networks on Pervasive Computing, 2000.

[36] J. Robert von Behren, Eric A. Brewer, Nikita Borisov, Michael Chen, Matt Welsh, Josh MacDonald, Jeremy Lau, Steve Gribble and David Culler. ***Ninja: A Framework for Network Services.*** University of California at Berkeley. 2002. (Also see the Ninija project in URL http://ninja.cs.berkeley.edu/relwork.html)

[37] Rekesh John. ***UPnP, Jini and Salutation - A look at some popular coordination frameworks for future networked devices.*** California Software Labs, June 1999. http://www.cswl.com/whiteppr/logincheck/genloginchek.asp?fname=upnppdf.zip

[38] Jan Newmarch. ***Jan Newmarch's Guide to JINI Technologies, Version 2.08.*** June 2001. URL http://jan.netcomp.monash.edu.au/java/jini/tutorial/Jini.xml

[39] Greg Butler, Lugang Xu, ***Cascaded refactoring for framework evolution***, Proceedings of 2001 Symposium on Software Reusability, ACM Press, 2001, pp. 51--57.

[40] Laurie A. Williams, Robert R. Kessler. ***All I Ever Needed to Know about Pair Programming I Learned in Kindergarten.*** 1999. URL http://www.agilealliance.com/articles/articles/Kindergarten.pdf. (Also see the articles in web site of Pair Programming. URL http://pairprogramming.com/)

[41] Larry L. Constantine, Lucy A. D. Lockwood. ***Software for Use: A Practical Guide to the Models and Methods of Usage Centered Design***. Addison-Wesley, 1999.

[42] Ben Shneiderman. ***Designing the User Interface (3rd edition)***. Pearson Addison Wesley, 1997

[43] G. Booch, J. Rumbaugh, I. Jacobson. ***The Unified Modeling Language User Guide***. Addison-Wesley, 1999.

[44] Martin Fowler, Kendall Scott, ***UML Distilled, A Brief Guide to the Standard Object Modeling Language***, Addison-Wesley. 2001.

[45] Simon Bennett, John Skelton, Ken Lunn, ***UML***, McGraw Hill, 2001.