

THE BIOAGENCY
A Collaborative Agent System for the
Discovery of Biological Information

by

Felix Munoz

A DISSERTATION

Presented to the

Division of Medical Informatics and Outcomes Research

and the Oregon Health Sciences University

School of Medicine

in partial fulfillment of

the requirements for the degree of

Master of Science

May 1999

Q171
M967
1999

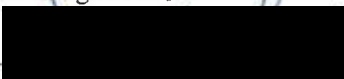
School of Medicine
Oregon Health Sciences University

CERTIFICATE OF APPROVAL

This is to certify that the M.S. thesis of
Felix Munoz
has been approved



Professor in charge of thesis  Dr. Christopher Dubay



Member Dr. Jingke Li




Member Dr. Kent Spackman



Member Dr. Jon Zonnana

Member



Associate Dean for Graduate Studies Dr. Richard Maurer

TABLE OF CONTENTS

TABLE OF CONTENTS	I
ACKNOWLEDGEMENTS.....	II
ABSTRACT.....	III
PREFACE	IV
I. INTRODUCTION	1
II. BACKGROUND – AGENT TECHNOLOGY	15
III. SYSTEM ARCHITECTURE	19
IV. AGENT ARCHITECTURE	28
A. <i>THE COMMUNICATIONS ASPECT</i>	34
1. <i>The Message</i>	34
2. <i>The Vocabulary</i>	41
B. <i>THE KNOWLEDGE MANAGEMENT ASPECT</i>	46
1. <i>The Knowledge Representation</i>	47
2. <i>The Knowledge Base</i>	50
C. <i>THE PLANNING ASPECT</i>	52
1. <i>Single-Agent Planning</i>	52
2. <i>Multi-Agent Planning</i>	54
V. LIMITATIONS.....	55
VI. IMPLEMENTATION.....	58
VII. CONCLUSION	62
APPENDIX 1 – THE PLANNING ALGORITHM	64
APPENDIX 2 – CLASS FAMILY	66
APPENDIX 3 – SAMPLE KNOWLEDGE.TXT	67
APPENDIX 4 – SAMPLE DICTIONARY.TXT.....	69
REFERENCES.....	70

ACKNOWLEDGEMENTS

I would like to thank Dr. Christopher Dubay, my advisor, for providing me with the freedom to pursue my own research interests and for providing me with the means to increase my knowledge in the areas I considered were relevant of study. His advice and support was priceless. I am very thankful. I would also like to thank Dr. William Hersh for giving me a chance to be a student in the Division and for providing the economic means to pursue my studies without worries. And finally, to my fellow members of the Zonana lab, Dr. Jonathan Zonana and Dr. Betsy Ferguson, for their advice and support on helping me realize my true calling. To them I am deeply indebted.

ABSTRACT

The research to be presented in this thesis addresses the topics of *information gathering*, *information extraction*, and *information filtering* in the area of bioinformatics. The thesis has two main goals. First, it will show the reader how the fluidity of the Web has made it cumbersome for scientists to use tools to access, manage, and analyze biological data. Second, it will show how collaborative agent technology can be used as a tool to solve this problem. The latter will be accomplished by presenting the reader with a software program developed specifically to provide bioinformaticians and scientists with a mechanism to delegate tasks to agents that are experts at performing specific services in behalf of the user.

PREFACE

As I started my research in the field of bioinformatics in the middle of 1997, I thought I had realized what it was that I could contribute to the field: user interfaces. By then there certainly were a lot of free services provided through the Web to the field of Medical Genetics. Many of them, however, were very obscure and quite difficult to use. A good user interface, I thought, can make any service, as complicated it may be, usable by a novice user.

With this in mind I began development of an application called “Research Team.” The goal of the application was simple: to provide, through a single application, a set of “Assistants,” each of which would be able to provide the user with an easy to use user interface to available services. After the development of the first Assistant, however, I began to see problems with this approach.

As I began development of an Assistant that was complementary to the first Assistant, I realized that, for the application to provide a better service, Assistants should be able to exchange information. With the system architecture I was working with at the time, the user would have to copy and paste information many times between the Assistants. A more efficient system architecture would allow an Assistant to use the same information the user had presented to another Assistant to perform its own tasks without requiring any user manipulation.

Another problem became apparent as I began development of the user interface for the second Assistant. Each Assistant provided its own user interface. At the rate of one user interface per service, the user would be inundated with user interfaces when attempting to perform complicated tasks. Additionally, there may be a learning curve that would need to be considered, as the user would have to spend time getting used to the new user interfaces. There had to be a better way.

An inspiration came one evening while watching "Star Trek: the Next Generation." I noticed the way that the crew interacted with their ship's main computer. "Computer," Captain Picard would say, "calculate the distance to the nearest solar system." To which the Computer promptly responded, "The distance to the nearest solar system is 2.3 light years." "Computer," Doctor Crusher would ask, "monitor the status of this patient and alert me when the pulse goes below normal." The Computer beeped approvingly and later woke her up with the alert.

This illustrates two completely different services performed without the need for either user to define the application that had to be used to fulfill the service that was needed. The ship's Computer provided the crew with a simple interface that allowed them to simply let it know what it was that was needed and the Computer would then call up the appropriate applications and fulfill the request. The convenience offered by this model was very appealing and I began researching the possibility of applying it to my "Research Team" system.

This thesis is a presentation of my attempt to provide the Medical Genetics community with a system that provides a user with a single, very simple to use interface for a set of services. I believe I have achieved my desired goals.

Felix Munoz

May 1999

I. Introduction

The research to be presented in this thesis addresses the topics of *information gathering*, *information extraction*, and *information filtering* in the area of bioinformatics. The thesis has two main goals. First, it will show the reader how the fluidity of the Web has made it cumbersome for scientists to use tools to access, manage, and analyze biological data. Second, it will show how collaborative agent technology can be used as a tool to solve this problem. The latter will be accomplished by presenting the reader with a software program developed specifically to provide bioinformaticians and scientists with a mechanism to delegate tasks to agents that are experts are performing specific services in behalf of the user.

The relevancy of this research becomes evident when one considers that is the nature of the scientist to manipulate and analyze data in order to generate new data. If the processes a scientist uses to access, manipulate, and analyze data become cumbersome, the scientist's ability to generate new data is slowed down.

Bioinformatics is an emerging interdisciplinary research area whose purpose is to provide scientists with tools that facilitate their ability to access, manipulate, analyze and share biological data. Because it deals with the management and analysis of a wide range of biological information, whether it relates to genes and their products, whole organisms, or even entire ecological systems, this field is viewed as an interface between the sciences of biology and information.

NCBI [Home](#) [Nucleotide Query](#) [BLAST](#) [Home](#) [?](#)

Other Formats: [FASTA](#) [Graphic](#)

Links: [Protein](#) [Related Sequences](#)

LOCUS R41NF1A5AA 1059 bp mRNA ROD 21-JAN-1992
 DEFINITION Rattus leucopus neurofibromatosis protein type I (NF1, type I splice variant) mRNA, partial cds.

ACCESSION M82634
 MID g205673

KEYWORDS neurofibromatosis protein type 1.

SOURCE Rattus leucopus (strain Fisher) fetus brain cDNA to mRNA.
 ORGANISM [Rattus leucopus](#)
 Eukaryotes; Mitochondrial eukaryotes; Metazoa; Chordata; Vertebrata; Eutheria; Rodentia; Sciurognatha; Synomorpha; Muridae; Murinae; Rattus.

REFERENCE 1 (bases 1 to 1059)
 AUTHORS Kyteais, A.P., Lee, P.S.Y., Hochizuki, H., Nishi, T., Levin, V.A. and Sava, H.
 TITLE Differential splicing of the neurofibromatosis type 1 (NF1) gene in rats: Homologous splice variants in human are expressed in rat cells
 JOURNAL Unpublished (1992)

FEATURES
 source Location/Qualifiers
 1..1059
 /organism="Rattus leucopus"
 /strain="Fisher"
 /db_xref="taxon:10115"
 /dev_stage="Fetus"
 /tissue_type="brain"
 gene 1..1059
 /gene="NF1"
 CDS 1..1059
 /partial
 /gene="NF1"
 /note="GAP-related domain (type I splice variant)."
 /codon_start=1
 /product="neurofibromatosis protein type 1"
 /db_xref="PIR:g205673"
 /translation="FHEVLTKILQOGTEPDTLAETVLADRFERLVELVTHWQDQGLP
 IARLANNVPCQNDLARLVLTLDSEHLTQLLHMFSEVELADSHQYLRHSL
 ASKIRITCFRNYGATYLRLLSPLRLIITSSQGVVFEVDPTLQPSSELSHGM
 LLQNTKFFHAISSSETFPSQLRVCCLYGVVQRFPOHIGAVGSAFLAFINPA
 IVSPYLAGLQKPPFRIERGLRSEKVLQSTAHNVLITREHERPFDVFCMSDLA
 RRFLEIASDCTSDAVHNSLFIIDGNVLAHLLHNNQETIQOVLSSNEDKAVQR
 RPFDRKATLALYLGPPENK"

BASE COUNT 269 a 246 c 285 g 269 t

ORIGIN
 1 tttatggag ccttgacaa aatcttcaa caaggacag aacttgatc acttgctga
 61 accgtgctt cagatcgtt cagagactt gtggaactg taacatgat gaaagacag
 121 gggagatc atatagatg ggtctggcc aatgtggcc ctggtatca gtgggatgg
 181 ctggatcag tctgggtcc gctgtctga tctggcatt cgtctacca gctgctcgg
 241 aacatgctt ctgaagcgt aggttggca gactctatg agactcttt tggagcaat
 301 agttggcca gcaagatca gactctctg tctagggtg aggtgctac ttaactcaa
 361 agatctctg accctctgt agactcatt atcaatcct cagattgga gattgtaga
 421 ctggaatg atcctaccg gctagagcc cctgagacc ctgagagaa caagagcac
 481 ctctctcga tgaagaaa gtctctcat gccatctca gttctcttc agagtctcc
 541 tgcagatc gaaagtctg caattgtct taccagctg ctgagcagc attctctag
 601 aacagatag gtgcagcgg aagtgcacg tctctgggt tcaatcaac tgcattgtr
 661 taccctcag agcagcgtt ttaagatca agcctcctc ctgagattg aagggcctg
 721 aagtctatg caaagctct cagagactt gccatctat taatctccc aagggagag
 781 aacatgagg atttaatga tctctgaaa agcaactcg acttggcag gatttttct
 841 ctgagatag caccagatg cctccagct gacagctaa accatgctt cctctctct
 901 agtgaatg atgtgtctg ttacatctg ctgcttggc ataatcagg gaaactggt
 961 cagtatctt ctgctcacc ggtctctaa gctctggaa gacgacttt agtcaagtg
 1021 gccacatc ctgactatc ggtctctcg gacagcagg

//

[View](#) the above report in [Macintosh](#) [Text](#) format

Figure 1. A sample DNA sequence record from the GenBank database. Notice the additional information available in addition to the sequence.

One of the branches of biology that has benefited tremendously from bioinformatics research is the field of *genetics*, which deals with the heredity and variation of organisms. By its nature, genetics researchers are forced to deal with the management and analysis of biological information on a daily basis. Whether it is population measurements, DNA sequences, lab outputs, etc., the genetics researcher is bombarded with biological information. The nature of the information the genetics researcher has to deal with is directly related to the type of research being performed. There are several types of genetics research. This thesis will concentrate on a specific type: the identification of gene function based on genetic sequence.

Any geneticist that has isolated a gene (or a fragment of one) is faced with the task of defining a function for the protein it encodes. Two complementary approaches have emerged to study gene function. *Functional genomics* emphasizes the roles of DNA and RNA in the natural progression from information (DNA) to function (protein). *Proteomics*, on the other hand, emphasizes the proteins themselves (William, 1997).

Because of the nature of the data required to define protein function using functional genomics techniques, this approach has received a lot of attention from the field of bioinformatics. While proteomics data is generally more graphical in nature (X-rays, 3D renderings, etc.), functional genomics data is mostly text based. DNA sequences can be represented as sequences of 4 possible letters. Proteins can be represented as sequences of 20 possible letters.

The emergence of the World Wide Web (WWW) provided bioinformatics researchers with a platform on which to develop tools to facilitate the sharing of genomic data. With these tools, genetics researchers had the ability to put their DNA and protein sequences into large databases, making them accessible to colleagues around the world. An example of such a database is GenBank, an annotated collection of all the publicly available DNA sequences. As of December of 1998, GenBank contained approximately 2,162,000,000 bases in 3,044,000 sequence records (Benson, 1998). Each GenBank record contains, in addition to the DNA sequence, information about the DNA fragment represented:

- A series of IDs useful for recalling the same sequence in several other databases.
- The organism from which the sequence was obtained.
- A reference to the paper that first mentioned the sequence.
- The list of features found in this DNA sequence.

One of the most common types of searches performed on the GenBank database is known as *homology searching*. In the following paragraphs, this bioinformatic task will be used as an example to help demonstrate how the current available tools, although adequate for the task, require too much time and effort to be used to their fullest extent.

Homology searching falls into the functional genomic analysis approach previously discussed. The general goal of performing a homology search on a DNA fragment is to retrieve a list of known DNA fragments that are similar to the DNA fragment being studied. The hope is that, by examining any functionality assigned to the proteins encoded in these homologous DNA fragments, some of the functionality of the DNA

fragment used for the search can be determined. For example, if the DNA fragment bears a lot of resemblance to a known DNA fragment that encodes a protein that breaks up complex sugars, it is very likely that the protein encoded in the DNA fragment in question may perform a similar function. Homology searching is usually the first step taken when a scientist isolates a DNA fragment. DNA fragments can be isolated in many different ways.

One of the *attributes* of a DNA fragment is its DNA sequence. A DNA fragment is made up of *nucleotides*. The fragment's DNA sequence represents the sequence of nucleotides that make it up. In DNA, there are only 4 possible nucleotides, represented by the letters A, C, G, and T. Therefore, it is customary to see a DNA sequence represented by a sequence of these letters:

ATGCGATGCAGTGAATTGCAGTGAAGT

If the DNA sequence belongs to a gene, it holds the code for the entirety or part of a protein. It can be *translated* to a sequence of *amino acids* (one amino acid for every 3 nucleotides), which is basically a representation of the protein.

Therefore, there are two general ways of doing homology searches when one has access to a DNA sequence. First, one can search for homologous DNA sequences by using the available DNA sequence directly. Second, one can search for homologous protein sequences by translating the DNA sequence into the corresponding amino acid

sequence first. Each approach has different advantages and disadvantages. For the sake of simplicity, however, we will concentrate in homology searches that use the available DNA sequence directly.

One of the most commonly used Web-based tools to perform homology searches on a DNA sequence is the “BCM Search Launcher”

(<http://kiwi.imgen.bcm.tmc.edu:8088/search-launcher/launcher.html>). This tool is a

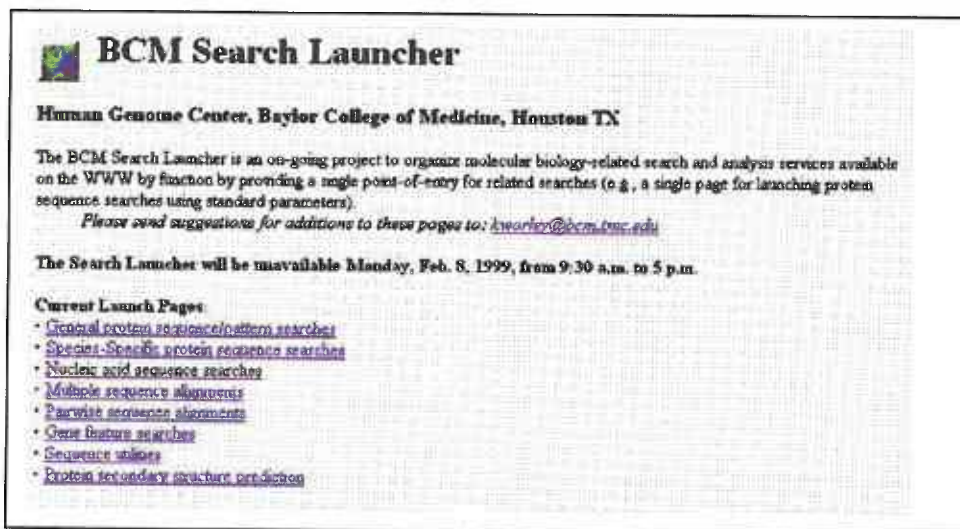


Figure 2. A section of the BCM Search Launcher web page, illustrating the variety of services provided by this site.

service provided by the Human Genome Center at the Baylor College of Medicine (Houston, Texas). The BCM Search Launcher is an integrated set of WWW pages that organize molecular biology-related search and analysis services available on the WWW by function, and provide a single point of entry for related searches (Smith et al., 1996). Of interest to us is the “Nucleic Acid Sequence Searches” set of services provided by the BCM Search Launcher (http://dot.imgen.bcm.tmc.edu:9331/seq-search/nucleic_acid-

search.html). As illustrated in Fig. 3, the Launcher provides access to a wide range of such services. The web page that the user is presented with, however, is not very user friendly. The names do not necessarily mean much to the novice user. In order to use this web page alone properly the user will have to become quite knowledgeable about each option available. We will concentrate our discussion on one of those services: the service named “BLASTN / nr dna - Gapped BLASTN with RepeatMasker, Entrez & SRS links (NCBI/UW/BCM).”

To the new user, the name of this service does not mean very much. In order to use this service, the user must first familiarize himself with several terms: “nr dna,” “RepeatMasker,” “Gapped BLASTN,” etc. Certainly, this is not asking too much from the user. After all, one needs to learn how to use an application if one intends to use it properly. Expand, this, however, to the 100 or so services provided in the BCM Search Launcher site alone. Admittedly, some of the knowledge gained from some services will also be useful in making use of other services. However, the point to be made here is that the user could be shielded from having to spend time learning a new user interface every time the need to use a different service rises.

BCM Search Launcher: Nucleic Acid Sequence Searches

Cut and paste DNA sequence here (raw sequences only, no header, etc.),
Maximum sequence length is 7000 bases.

Sequence name/identifier (optional):

Choose search method / database:

[H][O][P][E] = [H]Help/description, [O]Full Options search, [P]search Parameters, [E]Example search

- WU-BLASTX+BEAUTY / [see system](#) - Warren Gish's BLAST with gapped alignments, RepeatMasker, and BEAUTY post-processing (WU/BCM) [H] [O] [P] [E]
- BLASTX1.4+BEAUTY / [see system](#) - BLASTX 1.4 with RepeatMasker and BEAUTY post-processing that adds annotated domain information (NCBI/UW/BCM) [H] [O] [P] [E]
- BLASTN / [see data](#) - Gapped BLASTN with RepeatMasker, Entrez & SRS links (NCBI/UW/BCM) [H] [O] [P] [E]
- BLASTN / [blast](#) - Gapped BLASTN with RepeatMasker, Entrez & SRS links (NCBI/UW/BCM) [H] [O] [P] [E]
- TBLASTX / [blast](#) - 6-frame translation vs. 6-frame dbest
Gapped TBLASTX with RepeatMasker, Entrez & SRS links (NCBI/UW/BCM) [H] [O] [P] [E]
- BLASTN / [pmonth](#) - Gapped BLASTN with RepeatMasker, Entrez & SRS links (NCBI/UW/BCM) [H] [O] [P] [E]
- BLASTN / [blast](#) - Gapped BLASTN with RepeatMasker, Entrez & SRS links (NCBI/UW/BCM) [H] [O] [P] [E]
- BEAUTY-X / CRSeqAnnot - Seq family and domain information added to BLASTX search of BCM CRSeqAnnot (BCM) [H] [O] [P] [E]
- tRNAscan-SE - Search for tRNA genes in genomic sequences (BCM/WUSTL) [H] [O] [P] [E]

Rodent Specific Searches:

- BLASTN / [est mouse](#) - Gapped BLASTN with RepeatMasker, Entrez & SRS links (NCBI/UW/BCM) [H] [O] [P] [E]

Species Specific Searches:

- BLASTN / *Actinobacillus actinopterycomitans* genome - BLASTN 1.4 (OU) [H] [O] [P] [E]
- BLASTN / Fungal (*Aspergillus nidulans* and *Neurospora crassa*) genomes - BLASTN 1.4 (OU) [H] [O] [P] [E]
- BLASTN / *B. subtilis* - BLAST2 search of *B. subtilis* strain 168 (Pasteur) [H] [O] [P] [E]
- WU-BLASTN / *C. elegans* genomic - 6-frame translation vs. genomic (Sanger) [H] [O] [P] [E]
- WU-BLASTX / *C. elegans* wormpep - 6-frame translation vs. wormpep (Sanger) [H] [O] [P] [E]
- WU-TBLASTX / *C. elegans* EST - 6-frame translation vs. translated *C. elegans* EST (Sanger) [H] [O] [P] [E]
- BLASTN / *Cryptococcus neoformans* genome - BLASTN 1.4 (OU) [H] [O] [P] [E]
- BLASTN / *Neisseria gonorrhoeae* genome - BLASTN 1.4 (OU) [H] [O] [P] [E]
- WU-BLASTN / *S. cerevisiae* Genome - Warren Gish's BLAST with gapped alignments (Stanford SGD) [H] [O] [P] [E]
- WU-BLASTN / *S. pombe* genome - Warren Gish's BLAST with gapped alignments (Sanger) [H] [O] [P] [E]
- BLASTN / *Staphylococcus aureus* genome - BLASTN 1.4 (OU) [H] [O] [P] [E]
- BLASTN / *Streptococcus pyogenes* genome - BLASTN 1.4 (OU) [H] [O] [P] [E]
- BLASTN / *Streptococcus mutans* genome - BLASTN 1.4 (OU) [H] [O] [P] [E]
- BLASTN / *Synechocystis* sp. strain PCC 6803 Genome - (CyanoBase) [H] [O] [P] [E]
- WU-BLASTN / [ToxoDB](#) - WU-BLASTN search of the Toxoplasma database of clustered ESTs (UPENN) [H] [O] [P] [E]
- BLASTN / *Trypanosoma brucei* genome - BLASTN 1.4 search of *Trypanosoma brucei* BAC and P1 End (TIGR) [H] [O] [P] [E]

[Back to BCM Search Launcher Home Page](#)

Page Curator: Kim C. Worley, Human Genome Center, Baylor College of Medicine (e-mail: kim.worley@bcm.tmc.edu)
Last modified: Fri Mar 19 13:57:30 CDT 1999

Figure 3. The BCM Search Launcher page that provides the options for a nucleotide sequence homology search. Notice the names of the options.

What does this search do? BLASTN is an algorithm that compares a nucleotide query sequence against a nucleotide sequence database. The BLAST (Basic Local Alignment Search Tool) family of programs, of which BLASTN is part of, is a set of similarity search programs designed to explore all of the available sequence databases regardless of whether the query sequence is an amino acid sequence or a nucleotide sequence. The BLAST programs have been designed for speed, with a minimal sacrifice of sensitivity to distant sequence relationships. The scores assigned in a BLAST search have a well-defined statistical interpretation, making real matches easier to distinguish from random background hits. BLAST uses a heuristic algorithm which seeks local as opposed to global alignments and is therefore able to detect relationships among sequences which share only isolated regions of similarity (Altschul et al., 1990). For a discussion of basic issues in similarity searching of sequence databases, see Altschul et al. (1994). The Gapped BLAST algorithm is an extension of the original algorithm and it allows gaps (deletions and insertions) to be introduced into the alignments that are returned. Allowing gaps means that similar regions are not broken into several segments. This was one of the major shortcomings of the older BLAST algorithm. If two sequences had many non-neighbor homologous regions, the BLAST algorithm would report each region as a separate hit, leaving it to the user to piece the puzzle together. The new algorithm allows the presentation of these homologous regions as gapped alignments. The scoring of these gapped alignments tends to reflect biological relationships more closely (Altschul et al., 1997). To reduce the chances that the BLASTN algorithm reports homologies over one of the many repeats regions found in the human genome, the option we are using here uses RepeatMasker. This is a program that screens DNA sequences for interspersed repeats

NCBI BLAST Search with Entrez and SRS Links

BLAST search performed using the National Center for Biotechnology Information's [BLAST WWW Server](#)

Links to [Entrez](#) to the [Sequence Retrieval System \(SRS\)](#) and to [RepeatMasker](#) provided by the [Human Genome Center, Baylor College of Medicine](#)

Choose a program to use and database to search:

Program Database Perform ungapped alignment

The query sequence is [filtered](#) for low complexity regions by default.

Enter the sequence here in [FASTA](#) format:

Advanced options for the BLAST server:

Expect

Choose an organism from the [taxonomy database at NCBI](#) list to limit your BLAST search:

Genetic Codes (blastsr only)

Matrix with [Gap existence cost](#) and [Per residue gap cost](#) and [Lambda ratio](#)

Sequence Filtering:

Protein Filter

DNA Filter

More Options:

[Graphical Overview](#) [NCBI-GE](#)

Descriptions and Alignments

Other advanced options:

Mon Jan 26 15:23:37 CST 1993

Credits to: Tom Madden (madden@ncbi.nlm.nih.gov), Sergei B. Shavrin (shavrin@ncbi.nlm.nih.gov), and Jinghui Zhang (ejing@ncbi.nlm.nih.gov)

BCM Post-processing: [Ken Wierley](#) and [Pam Culberson](#)

Figure 4. BLASTN options.

known to exist in mammalian genomes as well as for low complexity DNA sequences.

The output of the program is a detailed annotation of the repeats that are present in the

query sequence as well as a modified version of the query sequence in which all the annotated repeats have been masked (replaced by Ns). This modified version of the sequence is then used in the BLASTN algorithm. On average, over 40% of a human genomic DNA sequence is masked by the program (Smit AFA & Green P, University of Washington, unpublished data).



Figure 5. Zoomed out view of a sample result set returned from a BLASTN search.

As we can see from Figure 4., BLASTN offers a variety of other parameters that can be defined by the user. For a description of each of these parameters, please refer to BLAST help page

(http://www.ncbi.nlm.nih.gov/BLAST/blast_help.html).

The BCM Search Launcher site provides two ways of performing a DNA sequence homology search: a default way and an advanced way. The default way simply requires the user to input the query DNA sequence in the

input box and to select a search method. The Launcher then performs the search using the default values for the parameters in Figure 4. The advanced way, on the other hand, presents the user with a way to modify the values for the search. Manipulation of these values should help the user to configure the application better for the specific type of

search that is being performed. This, however, is not necessarily an easy task. A lot of research needs to be done in order to understand what each parameter is and how it may affect the outcome. Given this, most researchers perform homology searches using the default values.

```
ERS gi|332996|gb|E20219|PP82CG Bovine papillomavirus type 2, complete genome.  
Length = 7937  
Score = 36.2 bits (18), Expect = 2.2  
Identities = 18/18 (100%), Positives = 18/18 (100%)  
  
Query: 32 atgcgatgcactgcaaatg 49  
      |||  
Sbjct: 7238 atgcgatgcactgcaaatg 7221
```

Figure 6. A homology from the result set returned after a BLASTN search.

Once a user has submitted a search, the returned result set is basically a long list of matches (see Figure 5). The number of matches reported can be very long (> 100 sometimes). From this result set the user can begin the process of determining which of the matches may hold information relevant to the DNA fragment being studied. Most commonly, this is done by investigating the information related to the DNA fragment from which the homologous DNA sequence was obtained. The BCM Search Launcher does provide a series of features that make this task easy. Figure 6 shows a sample homology, taken directly from the result set in Figure 5. As we can see, there are a lot of links available to the user. Both the “E” link and the link right next to the name of the DNA sequence take the user to the GenBank report of the DNA sequence, similar to the one in Figure 1. With this information one of the things the user can determine is if there

is any information related to the area of the sequence found to be homologous to the query sequence. From Figure 6, we can tell that the homology occurs over the area from nucleotide number 7221 through the nucleotide 7238. By looking at the “Features” section of the GenBank report, the user can see if there are any functionalities assigned to that region. If there is any, the user may have a lead to the possible functionality of the homologous area of the query sequence.



Figure 7. The “R” link.

Several other pieces of information can be gathered from the homology in Figure 6. The “R” link directs the user to a series of pages containing related sequences to the homologous sequence Figure 7 is one of such pages. Each of section of this page contains what is called a *neighbor sequence*, a sequence that shares a lot of similarities, sequence wise, to the homologous sequence. If neighbor sequences are very similar to the homologous sequence, it stands to reason that these sequences too may contain information useful to determining the function of the amino acid sequence (or protein) encoded in the query DNA sequence. Figure 7 shows one of these related sequences that come up from clicking on the “R” link. As one can see, there are several links the user can take to study this particular DNA sequence.

So, as we can see, there is quite a lot of information available to the user when performing a single homology search. There are several problems that can be identified from this description:

- *The amount of the information.* The number of homologous sequences can sometimes be quite large. Although usually the user limits himself or herself to analyze the best candidates (usually the top 10 homologous sequences), I believe this to be disadvantageous at certain times as they could be a very relevant homology that, because is too low in the ranking, is not even considered for examination.
- *The spread of the information.* As the user clicks on the links provided, the user is taken to the many sites that hold the information. Although this is not necessarily a bad thing, it does require more work from the user in learning how the user interface of each site works, what the vocabulary used in each site is, etc.
- *The dynamic nature of the information.* As more and more sequences are added to the databases, the homology search done today may be outdated tomorrow. New sequences may be added to the databases that may hold the key to discovering the function of the DNA fragment being studied. Furthermore, not only the information is dynamic, but its container is dynamic as well. Web sites with new services appear, disappear, and change routinely on the Web.

This list of problems drive to the conclusion that explosion of the Web has made it cumbersome for the scientist to access, analyze, and manipulate biological data. In order to solve this problem, new techniques for data management will need to be developed and implemented,

The rest of the thesis will discuss the use of collaborative agent technology as a tool to solve these problems. Section II of the thesis will first present the reader with an introduction to the field of agent and collaborative agent technology to familiarize the reader with the previous work done in this field. Section III will present the reader with the general architecture of the system developed. Section IV will discuss the inner workings of the agents that inhabit the system. And finally, Section V will address some of the limitations of the system.

II. Background – Agent Technology

Intelligent agents have been the subject of a great deal of speculation and controversy in the last few years. This has come to be because of two reasons: the growth of the Internet and the maturing of artificial intelligence (AI) research.

As the Internet has grown in the last few years, the need for mechanism to save people time by providing them with ways to delegate Web searches to software programs has become quite apparent. Agents are seen as user representatives that basically know, either by being told or by predicting, what it is that the user needs at any time and then providing that something to the user without the user having to be present.

The AI community's interest in agent technology is not necessarily linked to the Web. As the 1970's and 1980's came and went, the AI community has been trying to find the "killer app" to the AI techniques developed through several years of research in the computer science departments of universities. Techniques such as neural networks,

evolutionary programming, genetic algorithms, learning algorithms, etc. have matured in the AI research community but have failed to find a niche in mainstream computer environments. Agent technology has provided an opportunity for these techniques to be tried in a technology that has the potential of entering mainstream computer environments.

The concept of a software agent has been refined over the years. From HAL in Stanley Kubrik's *2001: A Space Odyssey* to the Computer in television's *Star Trek: The Next Generation*, the idea of a computer allowing us to ask it direct questions and obtaining an answer without the user having to perform the tasks themselves has fascinated people all over the world.

The *Webster's New World Dictionary* definition of an agent is "a person or thing that acts or is capable of, or is empowered to act, for another. This definition of agent is quite useful because of two reasons. First, it gives us a general understanding of what it is that an everyday person thinks what an agent is. It gives us a concept of what it is that a user expects when confronted with the term "agent." Second, it defines the two basic attributes of an agent: an agent does things, and an agent acts in behalf of someone or something. Translating this into computer terms, a software agent can be defined as:

A computer entity that performs user or another agent's delegated tasks autonomously

With this definition, let's review some of the previous research performed on the field of agent technology.

Perhaps one of the most well-known agents today, and arguably one of the most hated

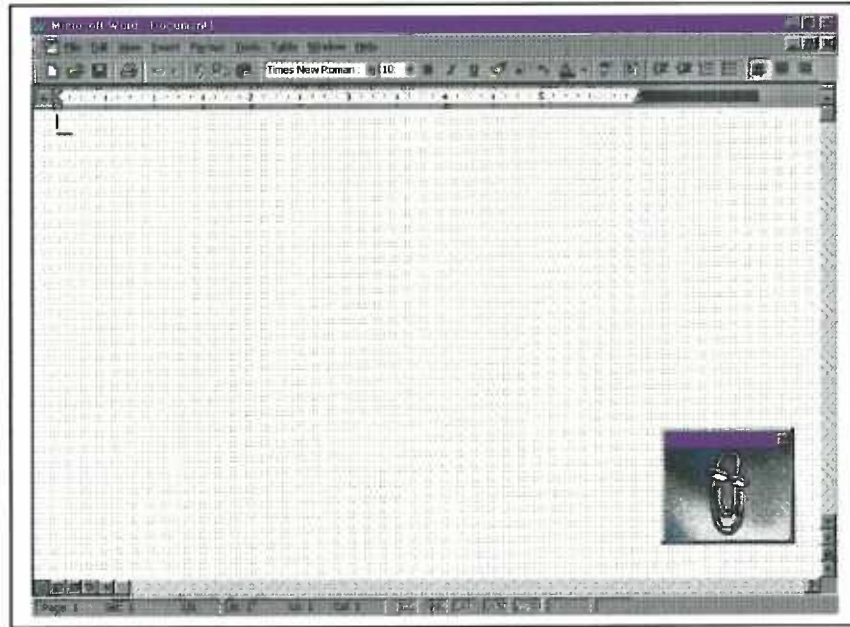


Figure 8. The Microsoft Office Assistant

ones, is the Microsoft Office Assistant (see Figure 8). This agent provides the user of the application with context-sensitive advice on how to perform certain tasks in the application. Unfortunately, one of its biggest flaws is that the agent provides this help without being requested by the user and it does so by interrupting the user from whatever it is doing and taking control of the application window. This behavior disturbs the interaction of the user with the application, and results in the user developing negative feelings against the agent.

Microsoft's Office Assistant represents the single-agent approach of agent technology. These agents are usually wizards that provide the user with a single service and perform these services much like a human expert might. These agents encapsulate knowledge.

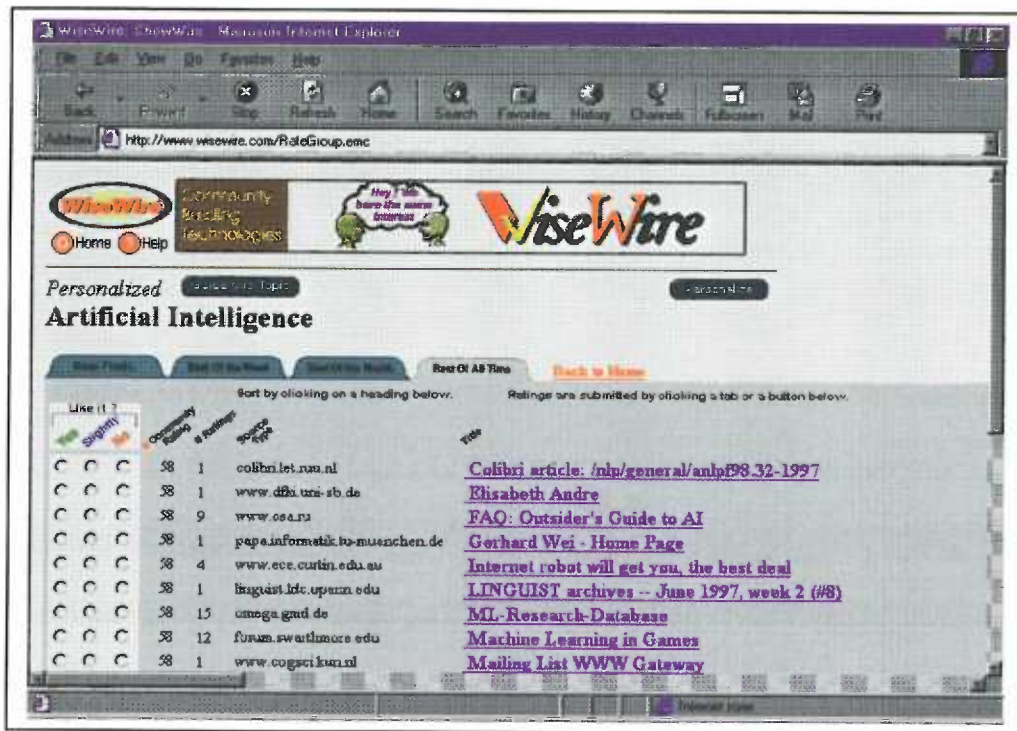


Figure 9. Multi-Agent System Example

This single service per agent model of single-agent technology, however, is a drawback for certain context. There are situations in which it would be preferable for agents to interact with each other in order to increase the quality of the output of the tasks they perform. This approach is called the multi-agent approach of agent technology. As an example of this technology, WiseWire is a company that has a Web-based product (see Figure 9) in which the user, after setting up its preferences, is able to have its personal agent select Web pages for the user to read that the user should find of interest.

The agent searches the Web basing its decisions on the likes and dislikes of the user and reports to the user only those pages that match the profile. However, the twist in this product is that personal agents from different users are allowed to “talk” to each other. While conversing, agents that have users with similar likes and dislikes are able to define sets of Web pages that another user has found interesting that another agent’s user may also find interesting because of their similar profile. This is an example of collaborating agents. Collaborating agents offer a very powerful mechanism by which the knowledge gathered by other agents can be used as a resource by other agent’s in the community that may have the need for the data but may not have the capabilities of accessing it.

This section has given a very broad overview of agent technology. For a very good reference of this exciting technology the reader is referred to the “Agent Sourcebook” (Caglayan, 1997). The next sections will introduce the reader to a system developed using collaborative multi-agent technology to facilitate the access of biological data and services to geneticists.

III. System Architecture

This section of the thesis will present the architecture that was constructed to support the design requirements presented previously.

During development of the system architecture, the primary design goal was to present the user with a very intuitive paradigm that was easy to relate to. The approach taken to achieve this was to base the system architecture on a real-life object. Since the

system involves agents, the metaphor of a travel agency seemed appropriate, since most people have had to deal with a travel agency and a travel agent at some point of their lives. This metaphor will be used in this section to shed some light into the reasons behind the system architecture developed. The architecture of the system will be presented by first defining the way in which a client would interact with a travel agency in order to purchase airline tickets, and then presenting the way in which a user would interact with the system in order to get the homologous DNA sequences of a given DNA sequence. By paralleling the way in which a person interacts with the travel agent and the way in which a user interacts with the system developed it should be simpler for the reader to understand how the system is supposed to work.

Before proceeding, let's first define the basic assumptions taken when designing the system. At the moment in which the user decides to use the system, the decision has already been made that the user would like to delegate a specific task to an agent. This defines two basic assumptions. First, the user has already formalized a task that needs to be delegated. Second, the user has formalized a set of information to be used to perform that task. In our examples, the tasks that need to be delegated are "purchase an airline ticket" and "get the homologous DNA sequences of a DNA sequence." The pieces of information related to the first task would be the date and the destination of the desired flight, while the piece of information related to the second task would be the sequence of the DNA sequence for which homologous DNA sequences. After establishing these two basic assumptions, the starting point for the user interaction with the system was defined.

When the task that needs to be delegated is to “purchase an airline ticket” one first calls the travel agency. Similarly, when the user wants to delegate the “get the homologous DNA sequences of a DNA sequence” to an agent, the user will first “call,” or startup, the Agency. As a side note, one of the benefits of working with agent technology (and object technology) is that it allows the developer the ability to give software components names that are easy to understand. In this case, for example, there is a piece of software actually called “Agency.”

Conceptually, one understands that an agency is where agents work. Furthermore, there are specific types of agencies, and hence specific types of agents that offer specific services. A travel agency is where one expects to find travel agents and, therefore, services related to traveling. A car insurance agency is where one expects to find car insurance agents and, therefore, services related to car insurance. In a similar manner, the system supports several types of Agencies. The Agency developed for this thesis is called the “BioAgency” and contains agents that provide biological services. There could be a second Agency, the “MedAgency,” which the user would expect to contain agents that offer services related to the medical field.

Once a person has called the travel agency, a receptionist usually answers the call. The receptionist’s job is to ask the person for his or her name, refer the person to a travel agent, and maybe keep track of whom the travel agents are handling. The receptionist, however, is not an agent. It represents a service of the travel agency. Within the system developed, this task is handled by a software component named “Receptionist.” This

software component, as discussed, is not an Agent. It is a service provided by the “Agency” software component. Its tasks constitute the user session management tasks of the system. When the user wants to initiate a session after starting up the Agency, the Agency initializes the Receptionist, which then takes the user’s name via a dialog box. The Receptionist then starts up a “User Agent,” and tells the User Agent the name of the user the Agent will be taking care of. The User Agent takes over the interaction with the user and, from the user’s point of view, the job of the Receptionist is completed. In the background, the Receptionist is also tasked with keeping track of the sessions that are currently active in the Agency, logging in the name of the user and the ID of the user’s User Agent. Whenever a user ends a session with the assigned User Agent, the User Agent notifies this action to the Receptionist, who removes the session from its active session list and either terminates the User Agent or defines the User Agent as available.

In a travel agency, if the customer is a repeat customer, the agency will more than likely have his profile. This profile should contain information about the customer, such as his name, address, phone number, previous travels, traveling preferences, etc. All of this information is retained in order to save the customer time by not requiring him to have to provide that information every time he calls the travel agency. If a profile does not exist, however, the travel agent would create a new profile for the customer. In a similar manner, in the Agency the User Agent has access to a User Profiles Database, where it keeps relevant information about every user that enters the Agency. This profile serves a purpose similar to the one kept in a travel agency. If a profile can not be found

for the user, the User Agent assumes the user is a new user and proceeds to create a new profile, which is then added to the database.

The travel agent is the person that should be able to take care of the client's request. Even though the task of purchasing airline tickets involves dealing with the billing department, a life insurance agent, etc., the customer does not expect to be referred to do all of these tasks. The travel agent is a one-point stop for all of the customer's needs. Furthermore, the travel agent's job is to find a way to satisfy the customer requests asking the customer as few questions as possible. The travel agent interacts with the necessary databases, talks with the billing agent to obtain a bill for the customer, talks to the insurance agent to set up life insurance for the flight, etc. In the system developed here, the User Agent has a similar job. After the login process is completed, the user interacts only with the User Agent. This way the user is completely shielded from what is going on in the background. The User Agent might make use of other Agents, of Web services, or other resources to fulfill the user's request. The only thing the user sees is the response to his request. The User Agent will attempt to find a way to fulfill the user's request and, if it is not possible to fulfill it, it will return an error message to the user with the reason why the request could not be fulfilled.

A travel agency usually has more than one travel agent. If a travel agent gets stuck trying to find out something for the customer, the other travel agents in the agency become resources. Different travel agents may have access to different resources. In the system developed, there are agents called, "Support Agents." These are Agents that

provide services that are helpful for fulfilling user requests. The OMIM Agent, for example, may provide access to information from the Online Mendelian Inheritance in Man (OMIM) database (NCBI, 1998). The Entrez Agent may provide access to information from the databases supported by the Entrez mega-database. In the present system, these agents do not interact directly with the user. The user makes use of their services by making requests to the user Agent, who then is tasked to forward those requests to the appropriate agent and to forward their responses to the user. This, however, is not a closed argument. Future user interfaces developed to access the Agency may permit the user to request tasks directly from specific agents. There is nothing in the system that prohibits this from happening.

Let's imagine now that the travel agency we have been talking about is a very big agency. The travel agents would probably not know each other, or if they do, they may not be familiar with what the other travel agents know. In this situation, when a travel agent needs information he doesn't have, he probably would not go from agent to agent soliciting information. He would probably first go to the one person in the travel agency that knows the most, or that at least has access to information about the agents working in the agency. This person would be the manager. Similarly, in the system developed, a User Agent does not need to know the names of all the support Agents available and the services they provide. This task is given to the Management Agent, also called the Manager. When an Agent begins its life in the Agency, it tells the Manager its name and describes the services it provides. With this information, the Manager can help other agents in the agency to find agents that fulfill specific services that the requesting agent

needs to fulfill a request. Let's say, for example, that the user wants to get the diseases related to a gene that has been found to be homologous to a DNA sequence. To do this the user provides the User Agent the name of the DNA sequence, which is all the information the user knows about the specific DNA sequence for which homologous DNA sequences are needed. After providing this information to the User Agent, the User Agent finds out that it alone can not fulfill the request. Therefore, it asks the Manager to provide a plan to fulfill the request. The Manager, knowing what every agent can do, is able to create a plan for the agent, which basically involves asking the appropriate agents specific requests that can be used to fulfill the request. In this case, the Manager determines that the BCM Agent is able to obtain the homologous DNA sequences of a DNA sequence. However, the BCM Agent needs the actual DNA sequence, not just the name. The Manager then looks if there is an agent that can provide the DNA sequence given its name and finds him (let's say the DNA Agent). Hence, the plan basically involves first asking the DNA Agent to get the actual DNA sequence of a DNA sequence given its name and then asking the BCM Agent to get the homologous DNA sequences of the given DNA sequence.

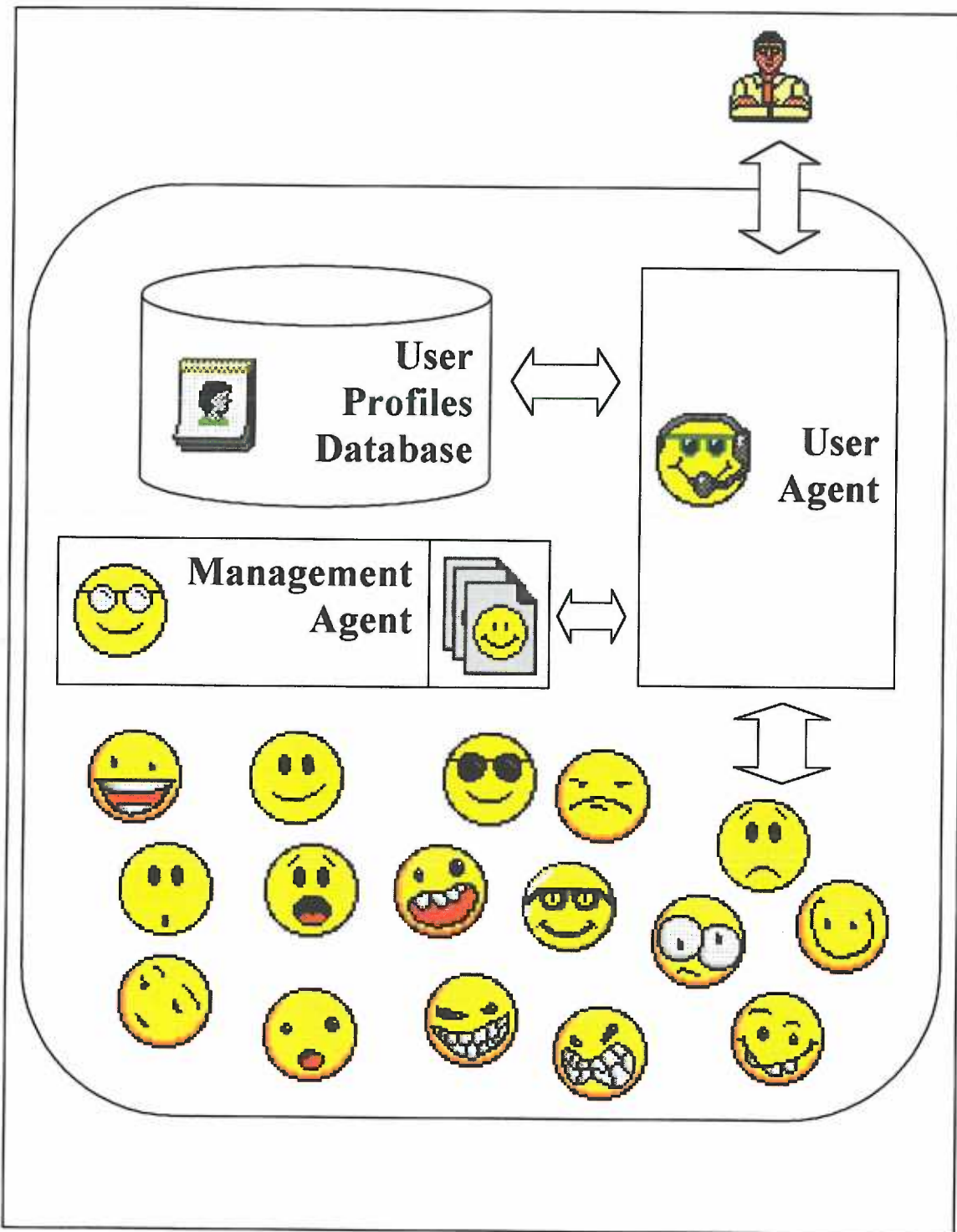


Figure 10. System architecture.

During the design of this architecture, the question came up whether agents should be able to remember the names of agents that provide specific services. Ideally, this would result in Agents not having to ask the Manager for Agent names every time the same request comes through. This ability, however, would severely limit the extensibility of the system. An example will help clarify this. Imagine that the User Agent has already asked the BCM Agent mentioned above for the homologous DNA sequences of a DNA sequence. It remembers that the BCM Agent can get this information, and that the DNA Agent can get the DNA sequence that the BCM Agent needs by providing it with the name of the DNA sequence. The next time that a similar user request comes in, the User Agent will no longer need to ask the Manager and will ask directly to the DNA Agent for the DNA sequence and pass this over to the BCM Agent which would then return the desired homologous sequences. What happens if the BCM-DNA Agent is introduced into the system? This agent is better than the BCM Agent or the DNA Agent alone, it has faster algorithms, and asking it the same question no longer requires two steps. Given that the User Agent is now circumventing the help from the Manager, the User Agent will never benefit again from the new Agent.

The system architecture presented here fulfills the system requirements presented in the introduction:

- *It is easily extensible*: the services provided by the system can be easily extended by simply adding new Agents into the system. The system would take care of facilitating the discovery of the services provided by the new Agents by centralizing the list of services in the Manager.

- *It is easily adaptable*: if the structure of the Web page an agent uses as a resource suddenly changes, the user needs only to upgrade the affected agent to a version that is aware of the page structure changes. The whole system does not become obsolete.
- *It limits the number of user interfaces the user must interact with*: the user will only need to interact with the user interface presented to him by the User Agent assigned to him.
- *It is customizable*: by allowing the User Agent to maintain a user profile of the user, the system is able to customize the user experience. At this point the only two things that are customizable are the type of user interface the user prefers to use and the user information the Agent can use when executing tasks in behalf of the user (the name, e-mail address, etc.).

This concludes this section of the thesis. Given the system architecture presented in this section, the Agents that inhabit this system will need to fulfill certain requirements.

The next section will discuss the Agent Architecture.

IV. Agent Architecture

This section of the thesis will introduce the reader to the internal make up of an agent. After reading this section the reader will be able to understand the basic behavior of an Agent and the mechanisms that it uses to help fulfill user requests.

As presented in the previous section, an Agent is basically an entity that provides itself and other agents with services that help fulfill user requests. These services are

presented to the community of agents through the Manager, which keeps a record of the services that each agent can do and provides agents with plans of action for specific requests.

As presented in the introduction, one of the primary system requirements was that it had to be easily extensible. Ease of extensibility, however, can be viewed in two ways. The first way is ease of extensibility as viewed by the user of the system which, as shown in the previous section, is fulfilled by the system architecture. The second way is ease of extensibility as viewed by a software developer. The system architecture provides the developer with a mechanism to easily deploy new agents and extend the Agency's capabilities but it does not provide with a mechanism to easily develop new Agents.

Before proceeding any further, it is important for the reader to understand the concept of *inheritance* as used in object-oriented (OO) programming. Inheritance, in this situation, refers to an object's ability to inherit attributes and methods (abilities) of another object. In its simplest form, inheritance in OO programming is achieved by *extending* existing objects. Let's say, for example, that a developer has programmed an object called *Person*. The developer has programmed into this object the ability to give its name, give its e-mail address, and give its telephone number. If the developer wants to create a new object, such as *Doctor*, that not only has the same abilities the *Person* object has but it also has the ability to give a diagnosis, the developer may decide to rewrite code to give a *Doctor* object to give its name, or *extend* the *Person* object. If the developer decides to extend the *Person* object, the *Doctor* object will have the same

attributes and abilities as the Person object by default and all the developer will need to program into the Doctor object is the ability to diagnose. This is clearly a very efficient way to reuse pre-existing code and saves the developer time.

The system presented here is written in a language called Java (Kramer, 1996). This language is an object-oriented language and, hence, provides developers with an inheritance mechanism. Given this, an approach to provide the developer with an easily extensible system was possible. All that is needed is to create a Basic Agent with the basic capabilities that every Agent in the Agency should have in order to interact with other agents and to perform their services. This would allow the developer of a new Agent to simply extend this Basic Agent and concentrate in programming the abilities needed to provide the services that the new Agent will offer.

With this as a general goal, it was necessary to define what exactly it was that the basic Agent should be able to do. Given the specific types of tasks the agents to be developed would be doing (i.e. homology searches) a specific set of requirements became apparent:

- *Agents must be able to dynamically communicate with each other.*

Agents should be able to advertise their services, request other Agents to perform services for them, and reply to request submitted by other Agents. Communication of this type, also called *dynamic communication*, allows the development of software where the software components do not have predefined communication behaviors. Most OO software allows communication between software components, but this

communication is, in essence, *hard-coded communication* between the components. For example, if there is Object A and Object B, the developer may define communication between these components by coding into Object A something like “ask Object B for the homologous DNA sequence of the DNA sequence.” This is called hard-coded communication because Object A can only ask to for the homologous DNA sequence of a DNA sequence from Object B. By using dynamic communication, however, Agents are able to do two things. First, they are able to change, on the run, who it is that they request services from. They may decide not to ask a specific Agent because it is too busy or because its results are not very reliable. Second, they are able to discover, on the run, new services added to the Agency by the addition of new Agents. Whenever a relevant service shows up on the Web all that needs to be done is to create an Agent that makes use of the new service, put it into the Agency, and the other Agents in the Agency will immediately be able to use this new service, without ever having to change one line of code on the older Agents.

- *Agents must be able to deal with concepts.*
- *A Nucleotide Sequence Homology Search is a concept.* It does not exist until one is actually performed. Concepts allow agents to communicate abstractly about services and the type of information they need to perform those services. For example, a BCM Search Launcher Agent can say that it can “get the *Homologous Nucleotide Sequences* of a *Nucleotide Sequence* given its *Value*.” The use of concepts also allows agents to resolve concept definitions by making use of IS-A relationships. Let’s assume, for example, that the BCM Launcher Agent has been requested to get the homologous nucleotide sequences of a *DNA Sequence*. However, in its service list the

Agent finds out that it can only get the homologous nucleotide sequences of a *Nucleotide Sequence*. The agent needs to be able to figure out, on its own, that a *DNA Sequence is a Nucleotide Sequence* and, therefore it is valid to use the provided *DNA Sequence* to perform the requested service.

- *Agents must be able to understand that a concept may have a set of related concepts.*

All instances of the *Nucleotide Sequence Homology Search* concept contain an instance of the *Parameters Set* concept and an instance of the *Result Set* concept.

Let's concentrate for a second around the *Result Set* concept. Figure 5 is the Web page that results from performing a *Nucleotide Sequence Homology Search* on the sequence:

```
AATCCGTAAGCTTTCATCGATCGATGGCATGATGCATGCACTGCAAATGGCCTAAGTCCATTGCAAAT
TGGAATTGAACCGGTTTTAACCCCTGAATTGACAAAGTCCAAAGTGGAACCTAAAGGTAAGTTGAACGG
TTAACTGGGTAAACGGTCAAATGAAACCTGGAACGTAAGTTGGGAATTGAACTGGAAAATTGA
```

This Web page contains the *Result Set* of the homology search. Disregarding the heading and the footer of the page, the main body of the Web page contains a series of very similar elements.

```

ERS g1|332996|gb|M20219|PP82CG Bovine papillomavirus type 2, complete genome
      Length = 7937

Score = 36.2 bits (18), Expect = 2.2
Identities = 18/18 (100%), Positives = 18/18 (100%)

Query: 32  atgcatgcactgcaaatg 49
          |||
Sbjct: 7238 atgcatgcactgcaaatg 7221

```

Figure 11 A sample homology resulting from performing a homology search.

Figure 11 shows a section of the results from a search for nucleotide homology using the BCM Search Launcher. The item shown in Figure 11 is an instance of the *Homology* concept. Every instance of a Homology has the following related concept instances:

- An instance of the *Homologous Sequence* concept.
- An instance of the *Expect Value* concept.
- An instance of the *Text Representation* concept.
- An instance of the *Homologous Region* concept.
- *Agents must be able to deal with concept instances.*

“AATCCGT...TTGA” is an *instance* of the *DNA Sequence* concept. It is an instance simply because things can be done to it, such as a *Nucleotide Sequence Homology Search*. Agents need to be able to distinguish what is a concept and what is the instance of a concept.

- *Agents must be able to remember concept instances.*

A homology search usually is not a one-time-only process. As more and more sequences are added to the genetic sequence databases, the homology search

performed today will need to be repeated tomorrow, or a few days from now. If the user wants an agent to repeat the homology search at a later date, the agent will need to remember two things: the *Parameter Set* used during the user-defined search, and the *Result Set* obtained from the previous searches. With the *Parameter Set* the agent can recreate the user-defined search, and with the *Result Set* the agent can determine which of the resulting *Homologies*, if any, are new and worth reporting to the user.

Given the above list of requirements, most of the basic services that an agent should be able to provide can be divided into three aspects: the communications aspect, the knowledge management aspect, and the planning aspect.

A. *The Communications Aspect*

While designing the mechanism by which an Agent communicates with other Agents, the goal was to provide the mechanism with an intuitive design. This becomes important also for the user because the User Agent interacts with the user in much the same way as it interacts with other Agents. Since most people have been exposed to e-mail messaging, this metaphor was extended to develop the communications aspect of the Agents. Hence, the communication unit in a conversation between two agents is the message. As it will be shown later, if the user will be allowed to communicate with Agents directly, the communication process will have the need for a vocabulary. This section will present the reader with these two basic components of the communication process.

1. *The Message*

In an e-mail message contains two clearly defined sections: the metadata of the e-mail message, and the content of the e-mail message.

a. The Metadata

The metadata section of an e-mail message consists of the set of information units that describes the message itself. It contains, among other data that we do not need to discuss here, three very important pieces of information: the sender's e-mail address, the receiver's e-mail address, and the title of the message.

While the e-mail address of the receiver is used to facilitate the routing of the message to its destination, the e-mail address of the sender of an e-mail message provides the receiver of the message with the information of who sent the message. With this information the receiver may apply qualitative data to the message (it is junk mail, it is important, etc.) or the receiver can use this information to reply to the e-mail message. Let's look at what this piece of information looks like:

munozf@ohsu.edu

An e-mail address is composed of two information units: the name of the user ("munozf") and the name of the e-mail server that takes care of receiving and sending e-mail messages for the user ("ohsu.edu"). The multi-agent system presented here has the need for an "e-mail address" of sorts to be setup for each agent. The Address of an Agent looks like this:

OMIMAgent@BioAgency

In the current implementation there is not really a need for the Address to include the name of the Agency in which the Agent resides. This feature of the Address has been added for future developments. One possible future development may be multiple agencies running within the same computer. Agents within each agency have their own IDs and each Agency can be recognized by a name. With an agent address such as the one presented, inter-agency communication would be possible. For example, one of the agents in the MedAgency, the Disease Agent, needs to get the OMIM ID of a disease. The Manager of the MedAgency, aware of the services provided by the Agents within its Agency, determines that there is no agent within its own Agency that is able to provide the OMIM ID of a disease. Before sending an error message, however, the Manager sends a request to the Managers in the other Agencies residing in the computer. The Manager in the BioAgency receives the request, determines that indeed there is an Agent within its Agency with the requested service (the OMIM agent), and sends this information over to the Manager in the MedAgency ("OMIMAgent@BioAgency").

The title of an e-mail message serves the purpose of an ID. It helps the people involved in the e-mail dialogue to recognize a thread of communication. For example, when an e-mail message is send titled "My request", the sender of the message can check for a reply by looking for a message in the inbox with the title "Re: My Request."

b. The Content

When one looks at an e-mail inbox, the e-mails there can be classified as belonging to one of three categories: a request, a reply, and junk mail. Junk mail is commonly ignored so it will not be addressed here. That leaves us with request-type messages and reply-type messages. Each of these types of messages has a very different type of content.

The Request Content

Request-type e-mail messages serve the purpose of informing the receiver of a task that the sender would like the receiver to perform. A requested task may look like “Can you please tell me the URL of that homology search site you found?” or “Can you please get the DNA sequence of the EDA gene?” Hence, the content of every request is a

Requested Task.

One of the goals when developing the communications aspect of the agents in this system was to endow the agents with the ability to exchange messages that are readable by the user. Even though the user may never need to see communication exchanges between Agents, it would be nice to provide the user with a user interface to the Agency which would allow the user to see the Agents in the Agency and see the messages that are being exchanged between them. Another reason was to allow the user to interact with the User Agent as if the user was an Agent. By allowing the request content to be written in a very dialog-like manner, the user would be able to send requests in the same way the Agents communicate. In order to achieve this, a very simple grammar had to be created.

Most tasks that are requested have a similar structure (or grammar). For example, taking the required niceties off the requested task,

Can you please get the DNA sequence of the EDA gene?

the stripped-down requested task looks something like:

Get the DNA sequence of the EDA gene

Studying the tasks that would Agents would need to perform in the BioAgency, a basic set of components became apparent:

- *The Requested Action*: this component defines the specific action that is expected from the Agent. In the above example that action would be “get.”
- *The Focus Attribute*: every action has a subject upon which the action will be performed. The focus attribute in the above example would be “DNA sequence”
- *The Focus Attribute Owner*: an attribute, by definition, belongs to something. In the above example the owner of the “DNA sequence” would be “EDA gene.” The “EDA gene” however, is an *instance* of the *Gene* concept. “EDA” is an instance of the *Name* concept, which is also an attribute of the *Gene* concept. Hence, a different way of saying “EDA gene” is “the gene with the name of ‘EDA.’” This is important, because the grammar should allow the Agent to define the concept the focus property owner belongs to and the values of the attributes that can be used to specify which

instance of that concept is being referred to. Hence, a grammar to define the focus attribute owner would look like:

THE + Concept Name + WITH THE + Attribute Name + OF + Attribute Value

Or, for example:

the gene with the name of "EDA"

The grammar of the focus attribute owner allows for multiple attributes:

the gene with the name of "EDA", the Entrez UID of "N1234", and the OMIM ID of
"MIM6768"

With all these three components defined, the request grammar can be defined in the following way:

Requested Action + THE + Focus Attribute + OF THE + Focus Attribute Owner

For example:

Get the DNA sequence of the gene with the name of "EDA"

With a grammar like this, a variety of the requests that agents would be exposed to are relatively easy to define:

- Memorize the DNA sequence with the name of “DNA-1” and the value of
“AATCCGTAAGCTTTCATCGATCGATGGCATGATGCATGCACTGCAAATGG
CCTAAGTCCATTGCAAATTGGAATTGAACCGGTTTTAACCTGAATTGAC
AAAGTCCAAAGTGGAACCTAAAGGTAAGTTGAACGGTAACTGGGGTAAC
GGTCAAATGAAACCTGGAACGTAAGTTGGGAATTGAACTGGAAAATTGA”
- Get the homologous DNA sequences of the DNA sequence with the name of “DNA-1”
- Memorize the name of the agent with the name of “User Agent”
- etc.

The Reply Content

When sending a reply to a request, the basic piece of information that needs to be transferred is what the request sender is expecting as a response. However, there are situations where you may not know the answer to the request and the reply simply contains a simple “Sorry, I don’t know that” or “Sorry, I don’t know how to do that.” Therefore, there are two types of replies: a “successful” reply and a “error” reply.

The successful reply does not need to contain anything else than the value that is expected by the request sender. If a person requests the name of a person in an e-mail message and the reply contains the string “Felix Munoz,” the person that has send the message knows that the string is the name of the person that was asked for. In the system,

the successful reply content contains only the values that were requested, although it does provide the receiver of the reply message with information as to what type of data is contained in the message. This way, the receiver of the reply message can verify if the data type of the reply message is the same as the expected data type.

The error reply contains only information about why the request could not be fulfilled. In the current implementation the error is not handled by the Agents in the system. The error is simply a developer-defined string such as “Could not find OMIM ID.” This information is simply send back to the request sender, which ultimately is presented to the user, which can then take action to resolve the error.

2. The Vocabulary

In order to not restrict the user to a predefined vocabulary, it became necessary to develop a mechanism for Agents to map terms that the user may prefer to use to refer to certain concepts to the preferred concept names that the Agents are able to deal with. There are two types of relationships that need to be supported by this mechanism: synonyms and plurals.

The use of synonyms is very common in genetic research. For example, “DNA sequence” can be referred to as “genetic sequence” or “nucleic acid sequence.” Hence, a requested task may take many different forms:

- Get the DNA sequence of the gene with the name of “EDA”
- Get the genetic sequence of the gene with the name of “EDA”
- Get the nucleic acid sequence of the gene with the name of “EDA”

To make things simpler for the Agents, it was decided to make use of a concept learned from the Unified Medical Language System (UMLS) (Selden, 1996). For each medical concept, UMLS defines a “preferred name.” It also defines “alternate names” for each concept, if they exist. Hence, an application that makes use of UMLS should be able to map alternate names to preferred names and create rules based only on preferred names.

In this system there is the concept of the “preferred name” and the concept of the “alternate name.” A synonym is an alternate name. Whenever the requested task contains a synonym:

Get the nucleic acid sequence of the gene with the name of “EDA”

the requested task can be transformed by the Agent into an standardized form previous to further processing:

Get the DNA sequence of the gene with the name of “EDA”

Allowing the use of plural forms of a concept name allows the construction of user-friendlier requested tasks. For example, the following sentence is well written:

Get the homologous DNA sequence of the DNA sequence with the name of DNA-1

The sentence, however, is not what a user would write to another person, as it is known that, more than likely, a DNA sequence has more than one homologous DNA sequence. A user would be more comfortable writing:

Get the homologous DNA sequences of the DNA sequence with the name of DNA-1

To the agent, however, the fact that the user is requesting “sequences” rather than “sequence” should not matter. The agent will report whatever homologous DNA sequences are found after performing the service with the given data, whether it is just one or many of them. Hence, as with the approach taken with synonyms, a plural is considered by agents an alternate name and a requested task that includes plurals will be converted to a standardized form containing the singular form of the preferred concept name.

A question that came up during development of this approach was whether to give this vocabulary ability only to User Agent, since it is the only one interacting with the user directly, and hence will be the only one being exposed to synonyms and plurals. It was decided against it, however. All the Agents in the Agency should have this ability in case a user interface is ever needed to be developed at a later time that would allow the user to interact with all the Agents in the Agency.

An Agent does not use the Vocabulary until it does not understand one of the concept names used in the query. Therefore, there should not be a runtime performance penalty

for having the Vocabulary in all the Agents. There is, however, a penalty during startup, since each Agent loaded into the Agency will need to load the Vocabulary during initialization.

The Vocabulary to be developed to support these two types of relationships in the agent system had to be easily extensible. A simple way to achieve this is to have the vocabulary stored in a plain text file. In order to increase the agent's vocabulary, the user needs only to add lines to this file. This could be done in any plain text editor, such as Notepad in Windows 95 systems, or SimpleText in Mac systems. Alternatively, new concept names, synonyms and plurals could be added programmatically through a user interface provided to the user for this purpose.

To make it easy for the user to add new alternate names to the Vocabulary, especially through any user interface that attempts to simulate a conversation with the User Agent, the grammar used in this text file had to be quite simple and very human readable. The following grammar was decided upon:

“(Related Concept Name)” + (Relation) + “(Preferred Concept Name)”

If a user wants to add synonyms or plurals to the Agents' vocabulary, all that needs to be done, before starting the Agency, is add lines such as the following to the “Vocabulary.txt” file:

“genetic sequence” is a synonym of “DNA sequence”

“genetic sequences” is the plural of “genetic sequence”

Adding definitions while the system is running poses a different challenge. Imagine the user has asked the User Agent:

Get the genetic sequence of the gene with the name of “EDA”

The User Agent would first attempt to identify the concept name “genetic sequence.” Not finding it in its knowledge, it assumes that the term is not a standardized name and hence looks in its Vocabulary. Not finding it there either, it sends the user the following error message:

I do not know what “genetic sequence” is.

The goal here, then, is to be able to tell the User Agent that the term “genetic sequence” is a synonym of “DNA sequence.” Since the user can communicate with the User Agent only through the use of requests, the idea is now to create a requested task that will allow the user to define what a “genetic sequence” is. The following example illustrates the request structure that was decided upon:

Remember the definition of the term with the name of “genetic sequence” and the definition of “is a synonym of “DNA sequence””

This sentence can be parsed and added to the Agent's runtime vocabulary and to the Vocabulary file. The user would, of course, have to repeat the initial request, but this time the User Agent would be able to map the term to a preferred concept name and transform the requested task in a standardized way.

B. The Knowledge Management Aspect

In order to support the knowledge services required by an agent, such as the ones described during the agent design portion of this section, an agent should be able to do the following:

- Load concepts
- Add and remove concepts
- Load concept instances
- Add, edit, and remove concept instances
- Define if a concept is a child of another concept. For example, define if the concept "DNA sequence" is a child of the concept "sequence."
- Define if a concept is a parent of another concept. For example, define if the concept "sequence" is a parent of the concept "DNA sequence."
- Define if a concept has a given attribute.
- Define if a concept instance with a given ID already exists in the knowledge base.

Hence, there are two separate aspects of knowledge management in an Agent: concept management and concept instance management. There are two complementary data

constructs that support these two aspects: the knowledge representation and the knowledge base.

1. The Knowledge Representation

A knowledge representation is a concise and unambiguous description of what principal entities are relevant in an application domain and how they can relate to each other. This set of entities is not a collection of facts that arise from an actual specific situation (as we will see later, this is what a knowledge base is), but it defines and provides all semantic entities and their potential interactions necessary to completely describe that situation (Schulze-Kremer, 1998). The entities described in a knowledge representation can be anything that the application has to deal with, such as objects, processes, functions, etc.

There is a set of constraints that a knowledge representation developer must follow (Schulze-Kremer, 1997):

- Each concept must be explicitly defined
- Each concept must be unambiguously accessible within the representation
- Each concept must be connected to one another by one or more relation links

The approach taken to provide all Agents with a knowledge representation ontology was very similar to the one taken to provide a vocabulary to them. There is a text file that contains the knowledge representation. This text file, like the one for the vocabulary, is easily updateable and can be modified during runtime.

The grammar used in this knowledge representation had the same objectives as the one developed for the vocabulary. It had to be human-readable and the parsing of it had to be unambiguous for the Agent.

The following is an example on how homologous DNA sequences are defined in the knowledge representation:

A sequence is a basic concept

A sequence has a name

A sequence has a value

A nucleotide sequence is a sequence

An RNA sequence is a nucleotide sequence

A DNA sequence is a nucleotide sequence

A DNA sequence may have one or more homologous DNA sequences

A homologous sequence is a sequence

A homologous DNA sequence is a homologous sequence

As we can see, there are several types of relationships that are used to describe a concept:

- IS A – used when defining that a concept is a child concept of another concept. This provides the system with an inheritance mechanism similar to the one previously discussed. The child concept “inherits” all of the attributes of the parent

concept, so these attributes do not need to be redefined for the child concept. For example, let's take "A DNA sequence is a sequence." Since we have already defined that a "Sequence has a name" there is no longer a need to define that "A DNA sequence has a name." This attribute is inherited from the sequence concept.

- **HAS A** – used when the concept has one and only one of the related concepts. These related concepts are usually attributes of the concept.
- **HAS ONE OR MORE** – used when a concept has multiple values for the same attribute. This relationship was added to provide the user with a more realistic way of representing attributes that have multiple values. It is much more natural for the user to write "A nucleotide sequence has one or more nucleotides" than "A nucleotide sequence has a nucleotide." One of the disadvantages of this approach, however, is that when the knowledge representation is being loaded and the string "may have one or more" is found in the sentence, the Agent is required to look at its Vocabulary to define the singular form of the concept name. However, this should not create a serious performance problem. As before, if a performance penalty exists, it will occur during startup of the application and not during runtime.
- **MAY HAVE A** – used when the related concept is optional in the concept. For example "A gene may have an OMIM ID."
- **MAY HAVE ONE OR MORE** – used when the concept has multiple optional attributes. The reasoning for this relationship is the same as the one for the "has one or more" relationship."

Most knowledge representation developers suggest that the concepts in an ontology should be shown in a tree pattern with their parent concepts (Schulze-Kremer, 1997). That is, for each concept there is one and only one parent concept. However, I believe this can be restrictive. There are several concepts that, depending on the context in which they are used, can be viewed as having different parent concepts. Let's take the concept of a "query sequence." To the "Sequence Agent," whose job may be to keep tabs on all the sequences that come through the Agency, a query sequence that was presented to the system by the user may be seen as a *Sequence*. To the "BCM Launcher Agent" the query sequence may be seen as a *Parameter* that can be used when performing a homology search. Hence, "query sequence" is a child of those two concepts. This is not a new idea, however. SNOMED RT allows multiple hierarchies for each concept (<http://www.snomed.org/rt2/sld004.htm>).

2. The Knowledge Base

If a knowledge representation describes concepts an Agent knows about, a knowledge base describes a list of concept instances an agent knows about. Within the knowledge base the Agent has access to a list of concept instances and the attributes that have already been defined for that instance.

The benefit of a mechanism such as this is that it allows Agents to "remember." A DNA sequence may have homologous DNA sequences. After a homology search, the current set of homologous DNA sequences can be remembered by the Agent so that, next time the homology search is performed, the list of new homologous DNA sequences can

be defined and post-processing can be performed only on these new sequences, saving the Agent valuable time.

As presented in the requirement list, Agents should be able to add, remove, and edit knowledge. An approach similar to the one taken with the vocabulary and the knowledge representation was taken. A text file, called "KnowledgeBase.txt" contains the list of instances all Agents should be aware of. As before, the grammar designed to add concept instances to this file is also human readable, making the knowledge base easily extensible. In order to refer to any concept instance, the Agent makes use of an ID that is defined for each concept instance.

The DNA sequence with the name of "EDA" has an OMIM ID of "OMIM1234", a value of "AATGACTGAT...AGCTACCTAG" and an Entrez ID of "E557"

If the user wants to define another attribute for this DNA sequence, the user may modify this sentence or, alternatively, a new sentence may be added to the file, either by using a text editor before the application is started or by adding it programmatically:

The DNA sequence with the name of "EDA" has a GenBank ID of "GB7890"

With this approach, the Agents are able to remember concept instances they encounter or are requested to remember by the user or another Agent. In the current implementation, all Agents share this knowledge base. In a future implementation a possible improvement to

this setup is to allow individual Agents to handle their own “personal” knowledge bases. This approach would allow Agents to be pulled out of the Agency and transferred to another Agency either in the same computer or in a different one while letting the Agent remember those concept instances it has added to its knowledge base.

C. The Planning Aspect

Requests sent to Agents will not always be sent in a form that can be directly executed. Requests may need to be processed and broken down into subtasks, and a mechanism needs to be implemented to allow the Agent to perform these steps non-redundantly.

In this system, there are two types of planning possible: single agent planning and multi-agent planning.

1. Single-Agent Planning

Single-agent planning in this system refers to the Agent’s ability to break up a task into subtasks it is capable of performing without help from other Agents in the system. This is the very first thing an Agent will attempt to do when presented with a request. For example, let’s imagine that the following two services are provided by the “OMIM Agent”:

- Get the OMIM ID of a gene given its name
- Get the DNA sequence of a gene given its OMIM ID

Given these two services, this agent should be capable of answering the request:

Get the DNA sequence of the gene with the name of “EDA”

Notice, however, that none of the two services can actually fulfill the request directly.

The Agent should be able to break up this task into the subtasks:

- Get the OMIM ID of the gene with the name of “EDA”
- Get the DNA sequence of the gene with the OMIM ID of “Step 1”

In order for this to be possible, the agent goes through the following analysis of its own services while defining a plan to resolve the request:

- Do any of my services provide the DNA sequence of a gene as a result? In this case the answer is yes.
- The Agent initializes a plan for the request and adds that service as a required step. If more than one service can provide the DNA sequence of a gene, then the Agent would initialize multiple plans, one for each service, and go through the present process with each of them. In this example, however, there is a single plan with the final step of “Get the DNA sequence of the gene with the OMIM ID of “Step 1”.
- What information does this service need to be able to obtain the “DNA sequence of a gene”? In this case, the single service needs the “OMIM ID of a gene.”
- Does the request or any of the existing steps in the plan provide the OMIM ID of the gene? At this point the answer is no.
- Do any of my services provide the “OMIM ID of a gene” as a result? The answer is yes.
- The Agent adds the step to the current plan.

- What information does this service need to be able to obtain the OMIM ID of a gene?
The service needs the name of a gene.
- Does the request or any of the steps in the current plan provides the name of a gene?
Since the request does provide the name of the gene, the answer is yes.
- The plan is complete.

More often than not, there will be multiple ways in which the Agent will be able to fulfill a request. This means that there is the possibility of Agents being able to generate more than one plan. When the Agent is presented with this situation, the Agent must decide which of the available plans is most appropriate to pursue. In the current implementation this is done by simply selecting the plan with the less number of steps. In future implementations the Agent may have the capability of deciding on the plan based on the approximate time each of the steps takes to be performed.

2. Multi-Agent Planning

One of the most powerful strengths of the system is the capability of Agents discovering services that other Agents have advertised in order to fulfill a request. This ability makes the request fulfillment process a very dynamic and powerful mechanism.

Every Agent in the Agency does not perform multi-agent planning. This is a specialized service provided by a single Agent, the Manager. As explained in the description of the system architecture, the Manager serves as a repository to all the services provided by all the agents. The Manager has direct access to this information and hence is the perfect candidate to perform this service.

Multi-agent planning does not differ too much from single-agent planning. As explained, in single-agent planning the Agent looks at the list of its own services and defines a plan of action based just on those plans. When the Manager is researching a plan of action, the set of services it looks into for services that fulfill the desired criteria is the list of services of all the Agents in the system.

In the case of multiple plans, future implementations will make use of this Agent's ability by allowing it to select plans in which multiple communications steps are not required (for example selecting those plans where the plan performer can fulfill most of the steps), or by allowing the Manager to make use of qualitative data about the Agents when recommending one in the plan. A plan provided by the Manager would look like this:

- Ask the ID Agent to get the OMIM ID of a gene given its name
- Ask the OMIM Agent to get the sequence of a gene given its OMIM ID

With this information, the Agent that requested the plan can start to perform the plan by sending out requests to the appropriate Agents. It is be the responsibility of the Agent that performs the plan to provide the performer of each step with the set of information the performer needs to fulfill the requests.

V. Limitations

At this point it is appropriate to define the set of limitations encountered during development:

- The request grammar is not entirely flexible. It requires the user to define a focus attribute for every action. There are requested tasks that may not have a need for a focus attribute. For example, when asking the Manager to load an Agent, the request would be a lot more natural if it was written as “Load the agent with the name of “OMIM Agent”.” However, at this point, the request has to be written as “Load the instance of the agent with the name of “OMIM Agent”.”
- The system needs to be able to use aliases for referring to Agents. This problem now occurs if two Agents of the same type need to be loaded at the same time. For example, a future version of the system may create instances of a specific Agent if the Agent is observed to be a very busy and its request queue is large. The system in this case would benefit from allowing the system to detect this and spawn another Agent of the same type. The reference to this Agent, however, cannot be the same as the Agent already in the Agency. Either a unique ID or an alias needs to be set for each Agent that loads into the Agency.
- If the Manager list of services of the loaded Agents is large, the process of defining a multi-agent plan has the potential of being very slow. The current implementation basically runs through an array of services looking for an appropriate service and does this as many times as it needs to complete the plan. This is wasteful, time wise. The Manager should be able to have a faster algorithm that will allow it to reject plans that do not have a hope of being completed earlier in the plan development process.
- The request syntax does not allow the description of two concepts in the same request. For example, it is not possible right now to say “Get the DNA sequence of

the gene with the disease named “EDA”. There are two concepts being described here and the request syntax does not allow this.

- The fact that the Manager is the only Agent in the system capable of developing multi-agent plans can become a performance bottleneck in a system where every agent needs a plan defined. This has the potential of hindering performance substantially. Another model that could be considered in a future implementation is a model in which, whenever an Agent needs an specific service, it broadcasts the need to all the Agents in the community and receives messages back from those that are able to offer the service.
- The Agents are highly dependent upon the Manager Agent. If this Agent crashes, the system will fail to perform any tasks that the User Agent is not capable of doing alone.
- The system does not deal with dead Agents. If an Agent crashes, it does not have a mechanism for telling the Manager to remove its services from the service list. The Manager will still have these services and will be recommending a dead Agent as a performer of a plan step.
- All Agents need to be on all the time. This is not a good approach if the computer where the system is running in is low in memory resources. A better approach is to provide the system with a way of allowing Agents to only be instantiated when they are needed and then go to sleep and release their memory once they have completed their designated task.
- User interfaces at this point are limited to simulations of the request-reply model of communication. Studies need to be done, , to evaluate other forms of user interaction,

and the underlying communication mechanism may need to be modified based on the results. One possible limitation of current user interfaces is that they are text based. Presentation of multimedia data has not been considered in this study. As the goal of this system was to provide a single, simple to use user interface, studies need to be done to demonstrate that added richness for user interactions are needed.

VI. Implementation

This section of the thesis will describe the current implementation of the system described in the previous two sections. It will also discuss some of the implementation issues discovered during development.

One of the first implementation issues encountered was the decision on the programming language that was to be used for development of the system. The programming language called Java was selected. There are several reasons for this decision:

- A program developed in Java (and following certain programming policies) should be able to run on any platform that supports the language without any modification. This has been one of the most powerful features of Java, and has helped the language in becoming one of the most popular languages in the Internet age. This feature is also required in the bio-agency. As in most environments that make use of computers on a day to day basis, the operating systems used in genetics laboratories are varied. With the system written in Java it can be used in the Windows 95/NT, Macintosh, and UNIX-Solaris computers commonly found in laboratories. Agents that are developed

in Windows 95 computers can be delivered to a user using the system on a Macintosh computer. This allows the deployment of the system in a larger user base.

- A program developed in Java can be developed to benefit from object-oriented programming techniques. Object-oriented programming allows for faster design and development, as the roles for Objects defined during design, such as Manager or User Agent, can be represented in the programming language, each contained within a single code module. The inheritance model provided by object-oriented programming languages such as Java saves the developer time by allowing the encapsulation of common methods in a single object that can then be extended multiple times into objects that share those methods without requiring the developer to have to implement those methods in every new object that is programmed.
- Java allows encapsulation of methods within objects. This feature allows Agents developed to be mobilized, either by floppy-disk or other network delivery methods, without requiring the user to do much other than find the file named the same as the object that needs to be transferred (such as "OMIMAgent.class"), put this file on a floppy, transfer it to another computer with the Agency system running, and load the Agent in the new system.

The Agency application presents the user with a user interface that attempts to provide an intuitive mechanism for logging in and starting sessions. The system supports multiple concurrent user sessions. At this point security mechanisms have not been established to allow different users to keep information away from other users of the

system. Future implementations should take this into consideration, as users may desire to keep certain information private.

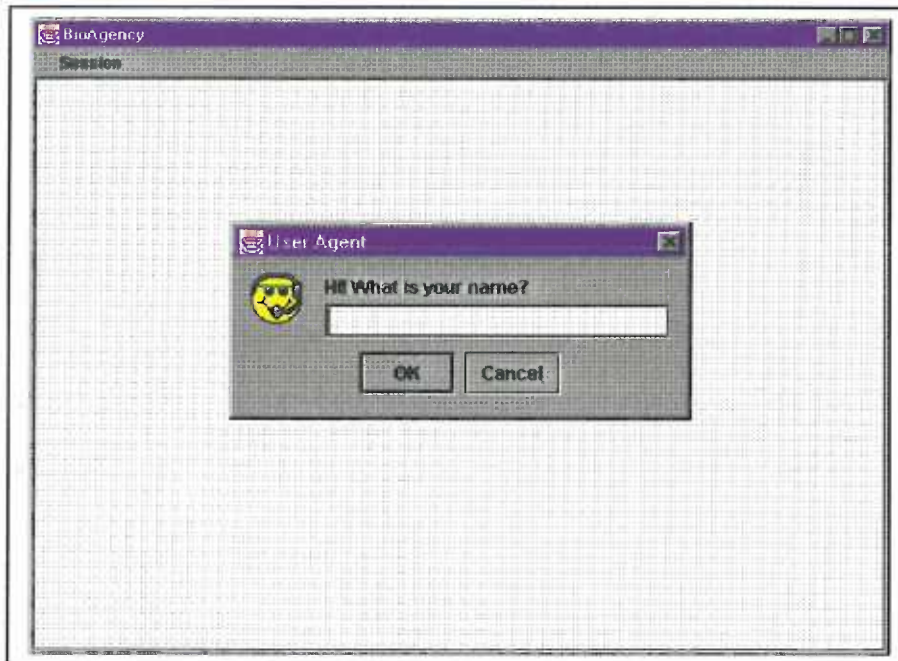


Figure 12. Agency User Interface

Another decision to be made was the type of user interface that needed to be developed to allow the user to have interaction with the User Agent. Given the metaphor being used, the most intuitive user interface was thought to be a chat-like interface. Such an interface allows the user to feel as if they are maintaining an ongoing conversation with another person who is able to perform the tasks that the user needs to be done. However, the system is not limited to this interface. The communication mechanism in the Agents does support e-mail like dialogues, where the user would have an inbox and an outbox, and the user would be able to send requests over to the User Agent via the outbox and receive replies via the inbox. The system does support multiple interfaces.

This allows the user to customize the system to use the type of interface that feels most comfortable. The User Agent can read this user preference from the user profile and present the appropriate user interface to the user.

Each Agent in the system runs as a separate thread. What this means is that each Agent does not have to wait for other Agents to answer to them to be able to reply to multiple requests. The Agents send requests to other Agents and, instead stopping their

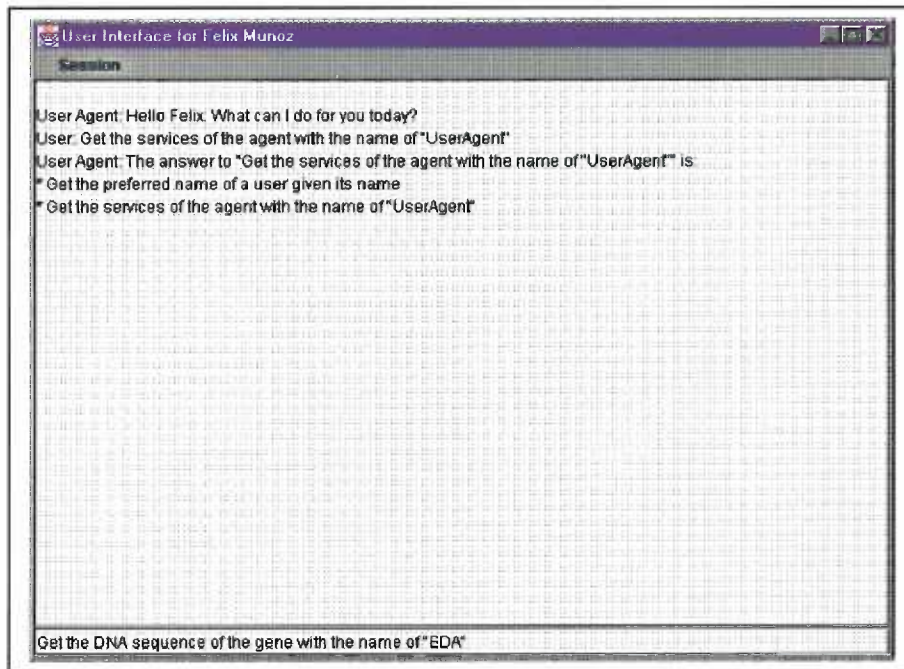


Figure 13. Chat User Interface

thread and waiting for a response, they proceed to process other requests while the other Agents, at the same time, are working on solving the request.

The Bioagency is basically an implementation of the system and agent architectures presented in previous sections. The system implementation fulfills the set of requirements presented previously. It also suffers from the described limitations of the architectures.

VII. Conclusion

This thesis has presented a system developed to make it easier for a scientist to access biological and other data available through the Internet. A system such as this presents the user with a single interface from which to perform a variety of data retrieval tasks that would take significantly more work for the user to perform using the user interfaces presented by many Web sites.

The system is based on an agent paradigm. Because this paradigm is implemented in an object-oriented fashion, the architecture of the system is very easily extensible. Its service self-discovery mechanism allows multiple developers to develop new agents for the system concurrently, without requiring them to know in advance ways in which to have agents interact. The system provides the agents with ways in which to advertise their services, to request services to be performed, and to reply to those services.

Communication between agents is not hard-coded but dynamic. In future implementations agents can be given freedom to choose who they communicate with, perhaps based on qualitative data on performance in previous interactions.

Much is left to be done. The limitations presented in the limitations section of this paper addresses some of the issues that still need to be solved in order for this system to become more useful, and to improve its usability. The work presented in this thesis, however, does accomplish the goal of presenting a proof-of-concept for the use of a

multi-agent technology approach to solve the problem of information overload in the Internet age.

Appendix 1 – The Planning Algorithm

Given a request and a list of knowledge units related to the request

Given an empty plan list

For each service in the agent's service list

 If the focus attribute of the service equals that of the request AND if the action of the service equals that of the request

 Add a new plan to the plan list with the service as a defined step

 Else if the focus attribute of the service is a child of the focus attribute of the request AND if the action of the service equals that of the request

 Add a new plan to the plan list with the service as a defined step

Set "Compiling plans" to true

While "Compiling plans" is true

 Set "Finished compiling" to true

 If there is at least one plan in the plan list

 For each plan in the plan list

 If the plan is considered to be "Undefined"

 Set "Finished compiling" to false

 For each knowledge unit provided in the request

 Add the knowledge unit to the list of provided knowledge units

 For each step in the plan

 For each knowledge unit required by the step

 Add the knowledge unit to the list of required knowledge units

 If the action of service of the step is "Get"

 Add the knowledge unit to the list of provided knowledge units

 For each required knowledge unit

 Set "Required attribute is provided" to false

 Set "Required knowledge unit is defined" to false

 For each provided knowledge unit

 If the attribute of the required knowledge unit equals the attribute of the provided knowledge unit

 If the current knowledge unit is defined

 Set "Required knowledge units is defined" to true

 If the provided value equals the required value

 Set "Required attribute is provided" is true

 Else

 Set "Required attribute is provided" is true

 Else

 If the provided attribute is a child of the required attribute

 If the current knowledge unit is defined

 Set "Required knowledge units is defined" to true

 true

 If the provided value equals the required value

```

        Set "Required attribute is provided" is true
    Else
        Set "Required attribute is provided" is true
    If the required attribute is not provided
        If the required is defined
            Set the current plan as "Impossible"
        Else
            Add current knowledge unit to the list of needed
            knowledge units
    If the current plan is considered to be "Undefined"
        If the plan does not need any knowledge units
            Set the current plan as "Defined"
        Else
            For each needed knowledge unit
                For each service on the agent's service least
                    If the service is a "Get" service AND the service's
                    focus attribute equals that of the needed knowledge unit
                        Add the service to the list of relevant services
                    If the number of relevant services is 0
                        Set the plan as "Impossible"
                    Else if the number of relevant services is 1
                        Add the step to the plan
                    Else if the number of relevant services is more than 1
                        Add the first service to the current plan and create a
                        new plan for each of the other services in the list of
                        relevant services

        If the current plan is considered "Impossible"
            Remove the current plan from the plan list
    If "Finished compiling" is true
        "Finished compiling" is false
    Select the shorted plan

```

Appendix 2 – Class Family

New agents can be created by extending a single abstract Java class that provides all of the basic agent behavior.

Agent.java

This is the basic abstract agent. It provides behaviors such as:

- Handle request
- Handle reply
- Send message
- Receive message
- Define a plan
- Perform a plan
- Verify if a concept is a child of another concept

HTMLAgent.java

This is another abstract class that extends the basic Agent class. This class was created to provide a common set of methods that agent designers could use to be able to extract information from an HTML document. Methods in this class include:

- Get HTML document
- Get content between tags
- Get tag attribute value

EntrezAgent.java, **BCMSearchLauncherAgent.java** and **MedlineAgent** all extend the HTMLAgent class. This way, the developer needs only to worry on developing the methods that will provide the services the agent claims it can do.

Appendix 3 – Sample Knowledge.txt

A sequence is a basic concept

A sequence has a name

A sequence has a value

A nucleotide sequence is a sequence

An RNA sequence is a nucleotide sequence

A DNA sequence is a nucleotide sequence

A DNA sequence may have one or more homologous DNA sequences

A homologous sequence is a sequence

A homologous DNA sequence is a homologous sequence

An amino acid sequence is a sequence

A parameter is a basic concept

A process is a basic concept

A process may have one or more parameters

A homology search is a process

A query sequence is a sequence

A query sequence is a parameter

A value is a basic concept

A measurement is a basic concept

A measurement has a value

A statistical measurement is a measurement

An EXPECT value is a statistical measurement

A representation is a basic concept

A text representation is a representation

A homology is a basic concept

A homology has a homologous sequence

A homology has an EXPECT value

A homology has a text representation

A BLAST search is a homology search

Appendix 4 – Sample Dictionary.txt

"Recommend" is a synonym of "Get"

"Retrieve" is a synonym of "Get"

"Show me" is a synonym of "Get"

"GenBank ID" is a synonym of "GenBank accession number"

“genetic sequence” is a synonym of “DNA sequence”

“genetic sequences” is the plural of “genetic sequence”

“nucleic acid sequence” is a synonym of “DNA sequence”

“nucleic acid sequences” is the plural of “nucleic acid sequence”

“DNA sequences” is the plural of “DNA sequence”

“homologous DNA sequences” is the plural of “homologous DNA sequence”

REFERENCES

- Altschul SF, Gish W, Miller W, Myers EW, and Lipman DJ (1990) "Basic Local Alignment Search Tool" *Journal of Molecular Biology* 215 (3): 403-10.
- Altschul, SF, Boguski MS, Gish W, and Wootton JC (1994) "Issues in searching molecular sequence databases" *Nature Genetics* 6: 119-129.
- Altschul SF, Madden TL, Schäffer AA, Zhang J, Zhang Z, Miller W, and Lipman DJ (1997) "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs" *Nucleic Acids Research* 25: 3389-402.
- Benson DA, Boguski MS, Lipman DJ, Ostell J, Ouellette BF (1998) "GenBank" *Nucleic Acids Research* 26 (1): 1-7
- Caglayan A, Harrison C (1997) "Agent Sourcebook" Wiley Computer Publishing
- Kramer D (1996) "The Java Platform" Sun Microsystems White Paper World Wide Web URL: <http://www.javasoft.com/docs/white/platform/javaplatformTOC.doc.html>
- Online Mendelian Inheritance in Man, OMIM (TM). Center for Medical Genetics, Johns Hopkins University (Baltimore, MD) and National Center for Biotechnology Information, National Library of Medicine (Bethesda, MD), 1998. World Wide Web URL: <http://www.ncbi.nlm.nih.gov/omim/>
- Schulze-Kremer S (1997) "Adding Semantics to Genome Databases: Towards an Ontology for Molecular Biology" *Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology* 272-5.

- Schulze-Kremer S (1998) "ISMB-98 Ontology Tutorial: Why Ontologies for Molecular Biology" *Ontologies for Molecular Biology Tutorial at the Sixth International Conference on Intelligent Systems for Molecular Biology* 1-16
- Selden CR, Humphreys BL (1996) "Unified Medical Language System (UMLS)" *Current Bibliographies in Medicine* 96-8, National Library of Medicine
- Smith RF, Wiese BA, Wojzynski MK, Davison DB, and Worley KC (1996) "BCM Search Launcher-An Integrated Interface to Molecular Biology Data Base Search and Analysis Services Available on the World Wide Web" *Genome Research* 6:454-62
- Williams KL (1997) "Functional Genomics" *Bio-Radiations* 99:4-6

19 ISSUES 2371
TH
5/99 72094-13