BioMAS: a Multi-Agent System for Genomic Annotation*

Keith Decker Salim Khan Carl Schmidt Gang Situ Ravi Makkena
Dennis Michaud

March 27, 2002

Abstract

The explosive growth in genomic (and soon, expression and proteomic) data, exemplified by the Human Genome Project, is a fertile domain for the application of multi-agent information gathering technologies. Furthermore, hundreds of smaller-profile, yet still economically important organisms are being studied that require the efficient and inexpensive automated analysis tools that multi-agent approaches can provide. In this paper we discuss the use of DECAF, a multi-agent system toolkit based on RETSINA and TAEMS, to build reusable information gathering systems for bioinformatics. We will cover why bioinformatics is a classic application for information gathering, how DECAF supports it, and several extensions that support new analysis paths for genomic information.

1 Introduction

Massive amounts of raw data are currently being generated by biologists while sequencing organisms. Most of this raw data must be analyzed through the piecemeal application of various computer programs and hand-searches of various public web databases. Typically both the raw data and any valuable derived knowledge will remain generally unavailable except in published natural language texts such as journal articles. However, it is important to note that a tremendous amount of genetic material is *similar* from organism to organism, even when they are as outwardly different as a yeast, fruit fly, mouse, or human being. This means that if a biologist studying the yeast can figure out what a certain gene does—its *function*—that other biologists can at least guess that similar genes in other organisms play similar roles. Thus huge databases are being populated with sequence data and functional annotations [3]. All new sequences are routinely compared to known sequences for clues as to their functions.

Furthermore, the *methods* by which biologists hypothesize function other than by similarity become other important clues to the identification of gene function in new organism sequences. The "function" of a gene can refer to one or all of concepts such as the molecular/biochemical function of a gene product, how this is used in some larger biological process, and also where (in the cell, or in which tissue) the gene product does its work [33]. For example, biologists can get clues to gene function by:

- looking for certain patterns of amino acids (called *motifs* or larger *domains*) that indicate likely molecular/biochemical activity
- looking for certain patterns of amino acids (especially at the ends of a protein) that indicate where the gene product will be used

*This work was supported by the National Science Foundation under grants IIS-9812764, IIS-9733004 and BDI-0092336.

- looking for information about similar gene products (proteins), rather than just similar genes

A large amount of work in bioinformatics over the past ten years has gone into developing algorithms (pattern matching, statistical, and/or heuristic/knowledge-based) to support the work of hypothesizing gene function. Many of these are available to biologists in various implementations, and now many are available over the web. Meta-sites combine many published algorithms, and sites specialize in information about particular topics such as protein motifs.

From a computer science perspective, several problems have arisen, as we have described elsewhere [7]. To summarize, what we have is a large set of heterogeneous and dynamically changing databases, all of which have information to bring to bear on the biological problem of determining genomic function. We have biologists producing thousands of possible genes, for which functions must be hypothesized. For the case of all but the largest and well-funded sequencing projects, this must be done by hand by a single researcher and their students.

Multi-agent information gathering systems have a lot to contribute to these efforts. Several features make a multi-agent approach to this problem particularly attractive:

- information is available from many distinct locations
- information content is heterogeneous
- information content is constantly changing
- much of the annotation work for each gene can be done independently
- biologists wish to both make their findings widely available, yet retain control over the data
- new types of analysis and sources of data are appearing constantly

We have used DECAF, a multi-agent system toolkit based on RETSINA [32, 12, 8]: and TAEMS [11, 34], to construct a prototype multi-agent system for automated annotation and database storage of sequencing data for herpesviruses [7]. The resulting system eliminates tedious and always out-of-date hand analyses, makes the data and annotations available for other researchers (or agent systems), and provides a level of query processing beyond even some high-profile web sites.

Since that initial system, we have used the distributed, open nature of our multi-agent solution to expand the system in several ways that will make it useful for biologists studying more organisms, and in different ways. This paper will briefly describe our approach to information gathering, based on our work on RETSINA; the DECAF toolkit; our initial annotation system; and our new extensions for functional annotation, EST processing, and metabolic pathway reasoning.

2 The RETSINA model of information gathering

We view information gathering as a catch-all phrase indicating information retrieval, filtering, integration, analysis, and display. In particular, information gathering is done in domains where information (unique, redundant, or partially redundant) is available at many different locations and is constantly being changed or updated, with even new information sources appearing over time. Centralized access is not available from the information sources because they are being produced by different organizational entities, usually for different purposes than those of the information gathering user. Examples of information gathering domains

are financial information (evaluating, tracking, and managing a stock portfolio)[8, 12], military strategic information (integration of friendly troop movements, enemy observations, weather, satellite data, civilian communications)[19], and annotation of gene sequences (as discussed briefly in the introduction).

Solutions to the information gathering problem tend to draw on two lines of technologies: research on heterogeneous databases and research on multi-agent systems. Work on heterogeneous databases in both the database and AI communities brings to bear the concepts of ontologies, wrappers, mediators, materialization, and query planning. Work on multi-agent systems brings to the table a way to actually embody wrappers and mediators; ways to do query planning in real-time domains; ways to deal with the dynamic nature of the data and the data sources; ways to handle issues of efficient distributed computation and robustness; ways to deal with the organizational issues involved in distributed information problems.

We promote the use of the RETSINA multi-agent organization[1] for building information gathering systems. The RETSINA approach consists of three general classes of agents[32, 12]:

- Information Extraction Agents, which interact directly with external data sources, i.e. wrapping sensors, databases, web pages.
- Task Agents, which interact only with other agents to handle the bulk of the information processing tasks. These include both domain-dependent agents that take care of filtering, integration, and analysis; also domain-independent "middle agents" that take care of matchmaking, service brokering, and complex query planning.
- Interface Agents, that interact directly with the end user.

A surprisingly large number of these agents are all or mostly reusable [8], which contributes to faster and faster prototyping of information gathering systems. DECAF (described in the next section) provides an implementation of these middle agents, reusable agent classes, and other tools for building multi-agent information gathering systems.

2.1 Operating Model

Our abstract model of information gathering relies primarily on query processing as its basic action. Interface agents primarily allow users to make direct queries (e.g. "should I buy shares in this stock" or "show me the HVT-1 genes that contain a prenylation motif"), or indirect queries via some materialized data view (e.g. a live web page representing a user's stock portfolio or a "clickable plant" representing all the available information on arabidopsis thaliana). There are several difficulties that need to be overcome even with such a simple model, namely:

- How to deal with the fact that the answers to queries change over time
- How to deal with the fact that typically user queries cannot be answered by routing to a single source
- How to deal with the heterogeneity of the user's information model when compared to the models of any other agent in the system
- How to deal with queries that would typically return tremendous volume of answers

[1] We will usually use the word *organization* to indicate the structure of a collection of agents, and *architecture* to indicate the structure of the internals of a single agent.

- How to deal with secondary or meta-expectations of the user with regards to the speed or resource usage expected of the query
- How to deal with an open system where the very structure of the queries that can be created may in fact change over time

From the information extraction end, then, all sources can be treated as "databases", but this again brings up similar, related issues: dealing with change, relating the available information to some common information model, how new sources can be added dynamically to a large system. Issues unique to these agents include how to deal with the fact that many sources are not actually complete databases, and attempting to buffer or otherwise ameliorate robustness and access issues for web-accessible resources.

Task agents fit in by either providing direct query services (i.e. access to some indirect information that can only be derived from analysis of other data) or support mechanisms that deal with some of the difficulties mentioned earlier. For example, middle agents such as matchmakers allow new services to be advertised dynamically, and then accessed by interface agents. Brokers or other types of middle agent mediators can provide seamless robust, load-balanced access to services. Query planning itself can often be computationally expensive enough to be handled by separate task agents.

To summarize, we model an information gathering system as an extended distributed query processing system. In particular, the multi-agent implementation of such a system deals with these "extensions" over traditional database systems:

- Dynamic Information: Data or derived information that changes over time
- Open Systems: data or derived information sources come and go over time
- Secondary User Utility: users don't just expect an answer, but they often have expectations about the time it will take to get that answer or how many resources (e.g. money) to spend to achieve an answer of some characterization (quality, certainty, etc.)

The next section will describe our realization of this general model using DECAF.

3 DECAF

DECAF (Distributed, Environment-Centered Agent Framework) is a Java-based toolkit for creating multi-agent systems [17]. In particular, several tools have been developed specifically for prototyping information gathering systems. Also, the internal architecture of each DECAF agent has been designed much like an operating system—as a set of services for the "intelligent" (resource-efficient, adaptively-scheduled, soft real-time, objective-persistent) execution of agent actions. DECAF consists of a set of well defined control modules (initialization, dispatching, planning, scheduling, and execution, each in a separate, concurrent thread) that work in concert to control an agent's life cycle. There is one core task structure representation that is shared between all of the control modules. This has meant that even non-reusable domain-dependent agents can be developed more quickly than by the API approach where the programmer has to, in effect, create and orchestrate the agent's architecture as well as its domain-oriented agent actions. This section will first discuss the internal architecture of a generic DECAF agent, and then discuss the tools (such as middle agents, system debugging aids, and the information extraction agent shell) we have built to implement multi-agent information gathering systems.

3.1 The DECAF Internal Architecture

DECAF provides the necessary architectural services of a large-grained intelligent agent [12, 32]: communication, planning, scheduling, execution monitoring, coordination, and eventually learning and self-diagnosis [21]. This is essentially the internal "operating system" of a software agent, to which application programmers have strictly limited access. The overall internal architecture of DECAF is shown in Figure 1. These modules run **concurrently**, each in their own thread. Details of the DECAF implementation can be found elsewhere [17].

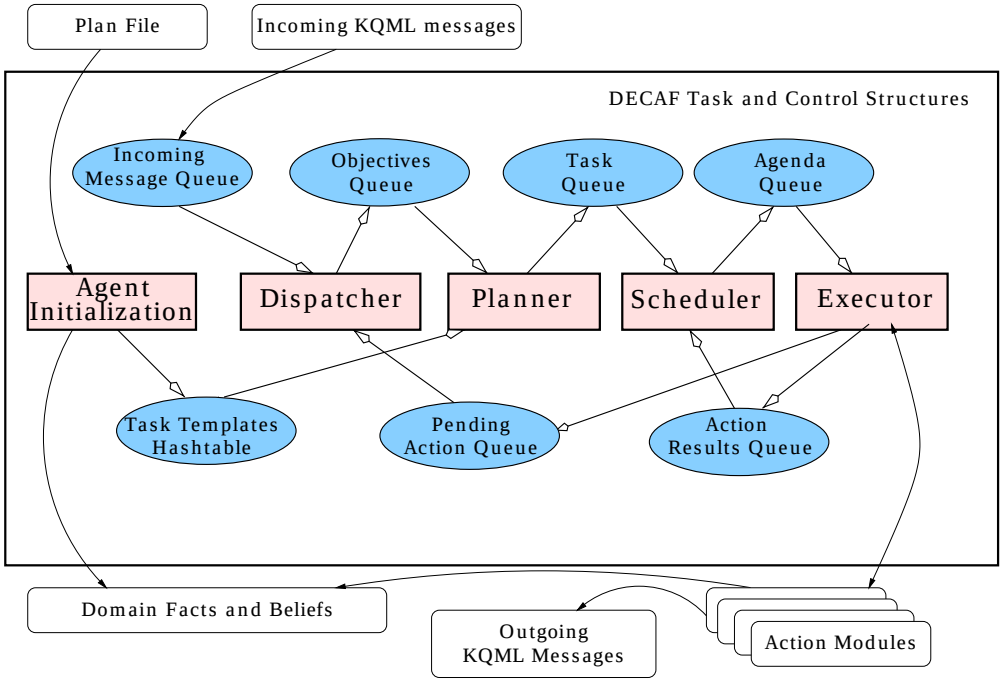

Figure 1: DECAF Architecture Overview

3.1.1 Agent Initialization

The execution modules control the flow of a task through its life time. After initialization, each module runs continuously and concurrently in its own Java thread. When an agent is started, the *Agent Initialization* module will run. The agent initialization module will read a *plan file* that describes the agent's capabilities as a specially-annotated HTN (Hierarchical Task Network). Each task reduction specified in the plan file will be added to the *Task Templates Hash table* (plan library) along with the tree structure that is used to specify actions that accomplish that objective.

3.1.2 Dispatcher

Agent initialization is done once and then control is passed to the Dispatcher which waits for an incoming KQML (or FIPA) message. These messages will then be placed on the *Incoming Message Queue*. An incoming message contains a KQML *performative* and its associated information. An incoming message can result in one of two actions by the dispatcher. First, the message may be a part of an ongoing conversation.

The Dispatcher makes this distinction mostly by recognizing the KQML `:in-reply-to` field designator, which indicates the message is part of an existing conversation. In this case the dispatcher will find the corresponding action in the *Pending Action Queue* and set up the tasks to continue the agent action.

Second, a message may indicate that it is part of a new conversation. This is the case whenever the message does not use the `:in-reply-to` field. If so a new *objective* is created (similar to the BDI "desires" concept[28]) and placed on the *Objectives Queue* for the Planner. An agent typically has many active objectives, not all of which may be achievable.

3.1.3 Planner

The Planner monitors the Objectives Queue and matches new goals to an existing task template as stored in the Plan Library. A copy of the instantiated plan, in the form of an HTN corresponding to that goal, is placed in the *Task Queue* area, along with a unique identifier and any provisions that were passed to the agent via the incoming message. If a subsequent message comes in requesting the same goal be accomplished, then another instantiation of the same plan template will be placed in the task networks with a new unique identifier. The Task Queue at any given moment will contain the instantiated plans/task structures (including all actions and subgoals) that should be completed in response to an incoming request.

3.1.4 Scheduler

The *Scheduler* waits until the Task Queue is non-empty. The purpose of the Scheduler is to determine which actions *can* be executed now, which *should* be executed now, and in what order they should be executed. This determination is currently based on whether all of the provisions for a particular module are available. Some provisions come from the incoming message and some provisions come as a result of other actions being completed. This means the Task Queue Structures are checked any time a provision becomes available to see which actions can be executed now.

It is possible to add significant reasoning ability to the scheduling module. This effort involves annotating the task structure with performance and scheduling information to allow the scheduler to select an "optimal" path for task completion.[2]

3.1.5 Executor

The *Executor* is set into operation when the Agenda Queue is non-empty. Once an action is placed on the queue the Executor immediately places the task into execution. One of two things can occur at this point: The action can complete normally (Note that "normal" completion may be returning an error or any other outcome) and the result is placed on the *Action Result Queue*. The framework waits for results and then distributes the result to downstream actions that may be waiting in the Task Queue. Once this is accomplished the Executor examines the Agenda queue to see if there is further work to be done. The Executor module will start each task in its own separate thread improving throughput and assisting the achievement of the real-time deadlines. Alternatively, an action may fail and not return, in which case the framework will indicate failure of the task to the requester.

[2] Optimal in this case may mean some definition of quality or deadline and real-time goals.

3.2 DECAF Task Structures

DECAF's underlying Hierarchical Task network (HTN) representation ties together two pieces of work: Williamson's work on information-flow representations used in RETSINA [37, 36], and Decker's work on representations of how local and non-local action executions effect those characteristics over which an agent expresses preferences (via a utility function) used in TÆMS [11, 34].

3.2.1 RETSINA Information Flow

The unique contribution of the RETSINA information flow representation used in DECAF is the declarative description of the information requirements of actions and the information producing abilities of actions [36]. This is in addition to the traditional precondition and effect representations used in planning systems. The information needs of an action are represented by a set of *provisions*. Provisions can be thought of as a generalization of plan action parameters and runtime variables, in which each provision has an associated *queue* of values. This information may be queued statically at plan-generation time or dynamically during plan execution. An action is *enabled* when there is at least one element queued for each of the actions provisions. Upon execution, the provision is consumed. *Parameters* are a subset of action provisions that are *not* consumed when an action runs (and thus do not involve a queue of values). When an action completes it produces both an *outcome* and a *result*. The outcome is one of a finite set of pre-designated symbols (e.g. the outcomes of CNLP or the observation labels of C-BURIDAN). The result is an arbitrary piece of information. *Provision Links* designate information flow of results from the outcomes of actions to the provisions of other actions.

3.2.2 TÆMS

TÆMS task structures are abstraction hierarchies whose leaves are instantiated basic actions or "executable methods". At a basic level this is similar to HTN (Hierarchical Task Network) or TCA (Task Control Architecture) approaches to action representation[16, 30]. Additionally, TÆMS allows the specification of dynamically changing and uncertain task characteristics that effect an agent's preferences (utility) for some state of the world, including tasks with hard or soft deadlines. A TÆMS specification also indicates relationships between local and non-local tasks or resources that effect these agent preference characteristics. Thus it extends HTN ideas toward specifying "worth-oriented" domains [29]. Recent extensions to TÆMS have included named provision relationships and multiple outcome specifications [36, 34]. In utility theory, agents have preferences over possible final states (action or plan outcomes), and preference-relevant features of an outcome are called *attributes*. A substantial body of work exists on relating attribute values to overall utilities [35]. At its core, TÆMS is about specifying these attributes and the processes by which they change—what we call a model of the task environment. In DECAF, we use TÆMS specifications to focus on three common attributes, *quality*, *cost*, and *duration*.

Actions. A DECAF *action* represents the smallest unit of analysis. For the purpose of utility calculation, each action has a probabilistic model, called the *behavior profile*, which specifies the likelihood of each outcome, and the probability distribution function for the quality, cost, and duration associated with each outcome.

Tasks. A DECAF *task* (or *subtask*) represents a set of related subtasks or actions, joined by a common *quality accumulation function*. For example, in an AND/OR tree, an AND task indicates that all subtasks must be accomplished to accomplish the task, while an OR task indicates that only one subtask needs to be accomplished. Since TÆMS is about worth-oriented environment modeling, it uses continuous rather

than logical quality accumulation functions (for example $\min$ instead of AND, $\max$ instead of OR[3]). For example, subtasks may be joined by a SUM quality accumulation function, indicating that as many subtasks as possible should be attempted. DECAF allows the explicit specification of a *characteristic accumulation function* for each characteristic (e.g. quality, cost, duration).

Plan Editor. The control or programming of DECAF agents is provided via an ASCII *Plan File* written in the DECAF programming language. The plan file is created using a GUI interface called the *Plan-Editor* which allows visual programming of HTNs and the TÆMS annotations. This provides a software component-style programming interface with desirable properties such as component reuse and some design-time error-checking. The chaining of activities can involve traditional looping and if-then-else constructs.

The DECAF Plan-Editor attaches to each action a performance profile which is then used and updated internally by DECAF to provide real-time local scheduling services. The reuse of common agent behaviors is thus increased because the execution of these behaviors does not depend only on the specific construction of the task network but also on the dynamic environment in which the agent is operating.

For example, a particular agent may be "persistent", or "flexible" [38] meaning the agent will attempt to achieve an objective, possibly via several approaches, until a result is achieved. This construction also allows for a certain level of non-determinism in the use of the agent action building blocks. Figure 2 shows a Plan-Editor session.

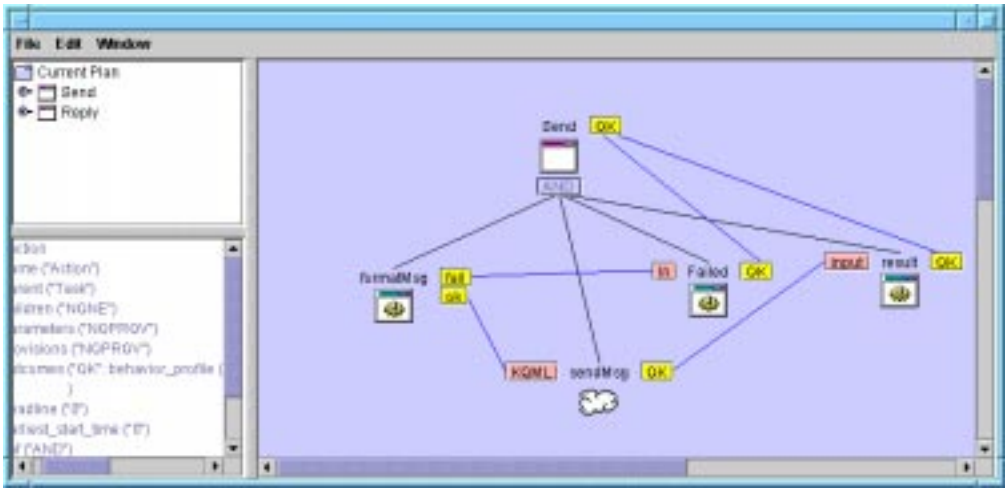

Figure 2: Sample Plan-Editor Session

3.3 DECAF Support for Info Gathering

DECAF provides core internal architectural support for secondary user utility. Thus DECAF plans can include alternatives, and these alternatives can be chosen dynamically at runtime depending on user constraints on answer timeliness or other resource constraints. DECAF also supports building information gathering systems by providing useful middle agents and a shell for quickly building information extraction agents for wrapping web sites. Agent name servers, matchmakers, brokers, and other middle-agents support the

[3]The full set of quality accumulation functions, including alternate definitions for AND and OR, is discussed in [10].

creation of open systems where elements may come and go over time. Dynamic information change is supported by reusable Information Extraction Agent behaviors that include the ability to push data values to the user, or to set up persistant queries that pull data from providers only when the answer changes significantly.

In order to support the development of agents, other tools have also been developed to support agent operations and software design. **Middle Agents** have been developed to support common multi-agent activities. A middle agent is an agent that facilitates agent operation while not directly related to completing a specific task.

3.3.1 Agent Name Server

The current DECAF Agent Name Server handles the registration of agents and "white-pages" services: mapping agent names to TCP addresses and port numbers. It is based on protocols in use at CMU's RETSINA project.

3.3.2 Matchmaker and Broker Agents

The Matchmaker agent serves as a "yellow pages" tool to assist agents in finding other agents in the community that may provide a useful service An agent will *advertise* its capabilities with the Matchmaker and if those capabilities change or are no longer available, the agent will *unadvertise*. The Matchmaker stores the capabilities in a local database. A requester wishing to ask a query will formulate the query to the Matchmaker and *ask* for a set of matching advertisements. The requester can then and make request directly to the provider. A requester can also *subscribe* to the Matchmaker and be informed when new services or interest are added or removed.

A *Broker* agent advertises summary capabilities built from all of the providers that have registered with one Broker. The Broker in turn advertises with the Matchmaker. For example, one Broker may have all the capabilities to build a house (plumber, electrician, framer, roofer, ...). The broker can now provide a larger service than any single provider can, and often manage a large group of agents more effectively [9].

3.3.3 Proxy Agent

DECAF agent can communicate with any object that uses the KQML or FIPA message construct. However, web browser applets cannot (due to security concerns) communicate directly with any machine except the applet's server. The solution is a *Proxy* agent. The Proxy agent is constructed as a DECAF agent and uses fixed addresses and socket communication to talk to Java applets or any application. Through the Proxy agent, applications outside the DECAF or KQML community have access to MAS Services.

3.3.4 Agent Management Agent

The Agent Management Agent (AMA) creates a graphical representation of agents which are currently registered with the ANS, as well as the communication between those agents. This allows the user to have a concept of the community in which an agent is operating as well as the capabilities of the agents and the interaction between agents. The AMA frequently queries the ANS to determine which agents are currently registered. These agents are then represented in a GUI. The AMA also queries the Matchmaker to retrieve a profile provided by each agent. This profile contains information about the services provided by an agent. This profile is accessible to the AMA user by double-clicking on the agent's icon. In the future, the AMA will also have the capability of monitoring and displaying communications between these agents. Each

agent will send a message to the AMA whenever it communicates with another agent, so that the user may then monitor all activity between agents.

3.3.5 Information Extraction Agent Shell

The main functions of an information extraction agent (IEA) are [8]: Fulfilling requests from external sources in response to a *one shot query* (e.g. "What is the price of IBM?"). Monitoring external sources for *periodic* information (e.g. "Give me the price of IBM every 30 minutes."). Monitoring sources for patterns, called *information monitoring* requests (e.g. "Notify me if the price of IBM goes below $50.")." These functions can be written in a general way so that the code can be shared for agents in any domain.

Since our IEA operates on the Web, the information gathered is from external information sources. The agent uses a set of *wrappers* and the wrapper induction algorithm STALKER [26], to extract relevant information from the web pages after being shown several marked-up examples. When the information is gathered it is stored in the local IEA "infobase" using Java wrappers on a PARKA [20] knowledgebase. This makes new IEA's fairly easy to create, and forces the difficult parts of this problem back on to KB ontology creation, rather than the production of tools to wrap web pages and dynamically answer queries. Currently, there are some proposals for XML-based page annotations which, if adopted, will make site wrapping easier syntactically (but still, does not solve the ontology problem—but see projects such as OIL).

4 A DECAF Multi-Agent System for Genomic Analysis

These tools can be put to use to create a prototype multi-agent system for various types of genomic analysis. In the prototype, we have chosen to simplify the query subsystem by materializing all annotations locally, thus removing the need for sophisticated query planning (e.g. [24]). This is a reasonable simplification since most of our work is with viruses that have fairly small genomes (around 100 genes for a herpesvirus and around 30 herpesviruses) or with larger organisms (e.g. chickens) for which we are constructing a consensus database explicitly.

Figure 3 shows an overview of the system as four overlapping multi-agent organizations. The first, *Basic Sequence Annotation*, is charged with integrating remote gene sequence annotations from various sources with the gene sequences at the Local KnowledgeBase Management Agent (LKBMA). The second, *Query*, allows complex queries on the LKBMAs via a web interface. The third, *Functional Annotation* is responsible for collecting information needed to make an informed guess as to the function of a gene, specifically using the three-part Gene Ontology [33]. The fourth organization, *EST Processing* enables the analysis of expressed sequence tags (ESTs) to produce gene sequences that can be annotated by the other organizations.

An important feature to note is that we are focusing on annotation and analysis services that are not organism specific. In this way, the resulting system can be used to build and query knowledgebases from several different organisms. The original subsystems (basic annotation and the simple query system) were built to annotate the newly sequenced Herpesvirus of Turkey (the bird), and then to compare it to the other known sequenced herpesviruses. Work is continuing to build a new knowledgebase from chicken ESTs.

4.1 Basic Sequence Annotation and Query Processing

Figure 4 shows the interaction details for the basic sequence annotation and query subsystems. We will describe the agents by their RETSINA classification.

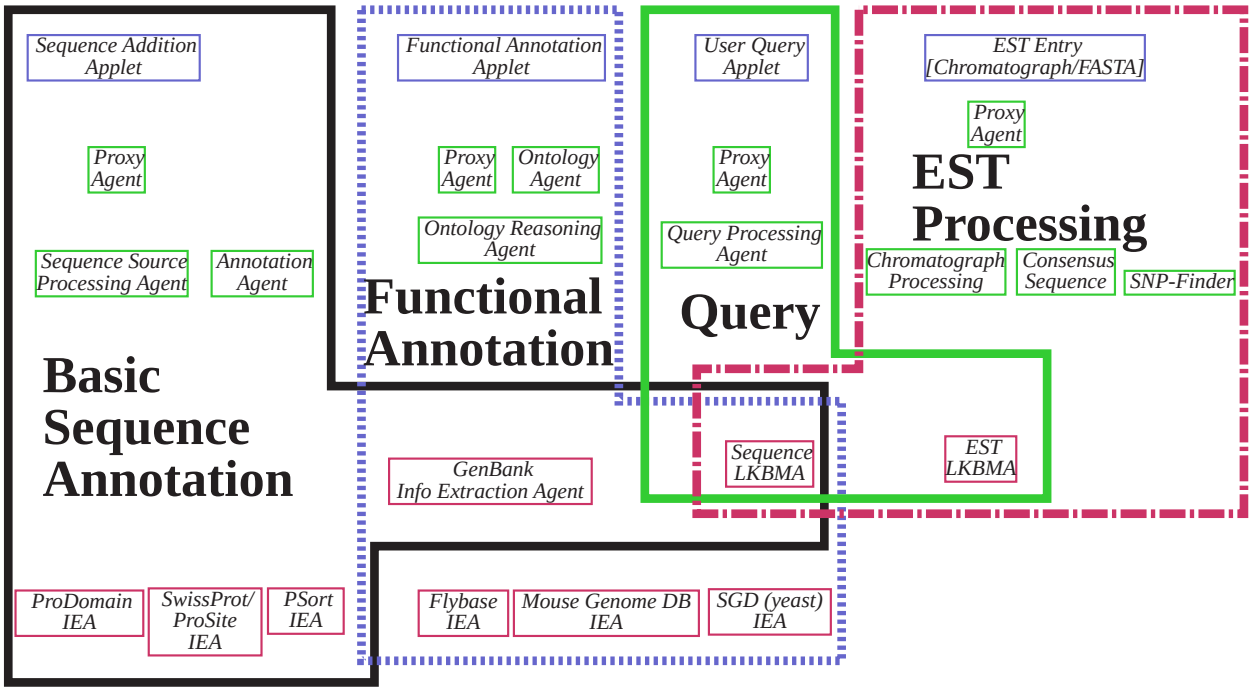

Figure 3: Overview of DECAF Multi-Agent System for Genomic Analysis

Information Extraction Agents. Currently 4 agents based on the IEA shell wrap public web sites. The Genbank wrapper primarily supplies "BLAST" services: given the sequence of a herpesvirus gene, what are the most similar genes known in the world (called "homologs")? The answer here can give the biologist a clue as to the possible function of a gene, and for any gene that the biologist does not know the function of, a change in the answer to this query might be significant. The SwissProt wrapper primary provides protein motif pattern searches. If we view a protein as a one-dimensional string of amino acids, then a motif is a regular expression matching part of the string that may indicate a particular kind of function for the protein (i.e. a prenylation motif indicates a place where the protein may be modified after translation by the addition of another group of molecules) The PSort wrapper accesses a knowledge-based system for estimating the likely sub-cellular location that a sequence's encoded protein will be used; in particular, estimating the number of "transmembrane domains" that indicate a protein that may "stick out of" the cell membrane, and thus be involved in cell signaling. The ProDomain wrapper allows access to other information about the encoded protein; a protein domain is similar to a motif but larger. As we move to new organisms, many more resources could be wrapped at this level (almost all biologists have a "favorite" here).

The local knowledgebase management agent (KBMA) is a slightly different member of this class because unlike most IEAs it actually stores data via agent messages rather than only querying external data sources. It is here that the annotations of the genetic information are materialized, and from which most queries are answered. Each KBMA is updated with raw sequencing data indirectly from a user sequence addition interface that is then automatically annotated under the control of an annotation task agent. KBMAs can be "owned" by different parties, and queried separately or together. In this way, researchers with limited computer knowledge can create sharable annotated sequence databases using the existing wrappers and other analysis tools as they are developed, without having to necessarily download and install them

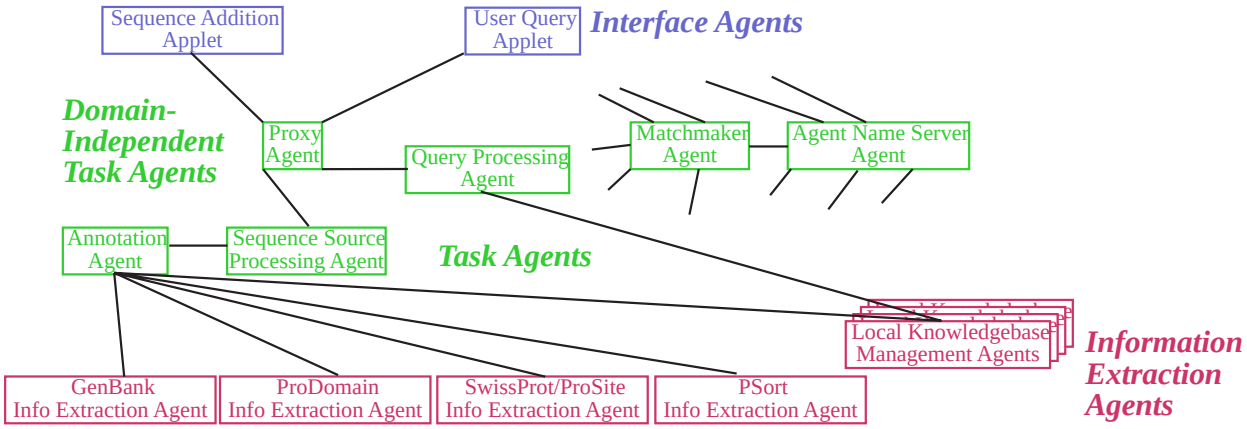

Figure 4: Basic Annotation and Query Agent Organizations

themselves. Using a PARKA-DB knowledgebase allows efficient, modern relational data storage on the back end and query as well as limited KB inferencing [20].

Task Agents. There are two domain task agents; the rest are generic middle agents described earlier. The Annotation Agent directs exactly what information should be annotated for each sequence. It is responsible for storing the raw sequence data, making queries to the various wrapped web sites, storing those annotations, and also indicating the provenance of the data (meta-information regarding where an annotation came from). The Sequence Source Processing Agent takes almost raw sequence data in ASN.1 format as output by typical sequence estimation programs or stored in Genbank. The main function of this agent is to test this input for internal consistency.

Interface Agents. There are two interface applets that communicate via the proxy agent with other agents in the system. One is oriented towards adding new sequences to a local knowledgebase (secured by a password) and the other allows anyone to query the complete annotated KB (or even multiple KBs). The interface hardly scratches the surface of the queries that are actually possible, but a big problem is that most biologists are not comfortable with complex query languages. Indeed, the simple interface that allows simple conjunctive and disjunctive queries over dynamic menus of annotations (constructed by the applet at runtime from the actual local KB) is quite advanced as compared to most of the existing public sites that allow textual keyword searches only. The current interface[4] allows the user to query based on gene name, gene product name, number of transmembrane domains, keyword search in homolog text descriptions, virus taxonomy, protien motif ontology, or any high-level term in the three GO ontologies.

For example, the table below shows a summary of the annotation information gathered by both the basic annotation agents, and the functional annotation agents (described next) for a representative gene product from gallid (Chicken) herpesvirus 3 gene SORF4. The input to the system was the gene product (translation) which is a string of 322 characters representing the amino acid sequence.

[4]Will be located at http://udgenome.ags.udel.edu; However, if reviewers wish to check the interface out the current interface demo is located at http://cgi.eecis.udel.edu/ makkena/cgi-bin/menupop/testpopups.html

Transmembrane Domains	1
Motifs (5)	ASN-Glycosylation Site (2) CAMP- and CGMP-Dependent Protein Kinase Phosphorylation Site (1) Protein Kinase C Phosphorylation Site (1) Casein Kinase II Phosphorylation Site (4) N-Myristoylation Site (3)
Domains (2)	Protein Glycoprotein Kinase Marek's Disease Virus (length 345; Diameter 91; PAM Radius of Gyration 36; PAM Sequence Closest to Consensus O89266_VVVVV 4-319; Distance 30 PAM) HYPOTHETICAL 32.1 KD Protein (length 294; NO CONSISTENCY VALUES)
BLAST hits (12)	R-SORF1 Protein [Gallid Herpesvirus 3] R-SORF1 Protein [Gallid Herpesvirus 3] SER/ARG-related Nuclear Matrix Protein; Plenty-Of-Prolines-10 [Mus Musculus] Extensin-like Protein [Zea Mays] AHM1 [Triticum Aestivum] Extensin [Volvox Carteri] Hypothetical Protein RV3876 [Mycobacterium Tuberculosis H37RV] HYP-rich Glycoprotein [Zea Diploperennis] Wiskott-Aldrich Syndrome Protein Homolog [Fission Yeast] Wiskott-Aldrich Syndrome Protein Homolog 1 [Fission Yeast] Strong Similarity to Unknown Protein (GB—AAD23008.1) [Arabidopsis Thaliana] Hydroxyproline-rich Glycoprotein [Perennial Teosinte] Kexin-like Protease KEX1 [Pneumocystis Carinii F. sp. Muris] Avena Neural Variant [Gallus Gallus] CG15021 Gene Product [Drosophila Melanogaster]
Biological Process	signal transduction [GO: 7165]
Molecular Function	two-component sensor molecule [GO: 155]
Cellular Component	membrane fraction [GO: 5624]

4.2 Functional Annotation

Figure 5 shows the Functional Annotation subsystem. This subsystem is responsible for assisting the biologist in the difficult problem of making functional annotations of each gene. Unfortunately, many of the millions of genes sequenced so far have fairly haphazard (from a computer scientist's perspective) functional annotation: simply free natural language descriptions. Recently, a fairly large group representing at least some of the primary organism databases have created a consortium dedicated to creating a gene ontology for annotating gene function in three basic areas: the biological process in which a gene plays a part, the molecular function of the gene product, and the cellular localization [33]. The subsystem described here supports the use of this ontology by biologists as sequences are added to the system, eventually leading to even more powerful analysis of the resulting KBs.

Information Extraction Agents. Besides the gene sequence LKBMA and the GenBank IEA, we are wrapping several organism-specific gene sequence DBs—for Drosophila (fruit fly), Mus (Mouse), Saccrynomaeces cervasie (yeast), and Caenorhabditis elegans (nematode). Each of these organisms is part of

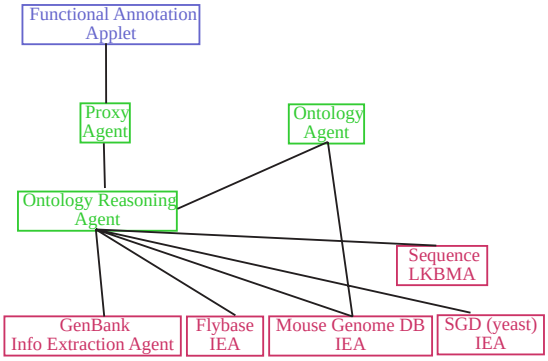

Figure 5: Functional Annotation Agent Organization

the Gene Ontology (GO) consortium, and has spent considerable time in making the proper functional annotation. With respect to the qualitative reliability of the annotations used by GO, each of these databases has a significant number of gene products that are human-annotated, as opposed to being Inferred by Electronic Annotation (also referred to by the acronym IEA). We are also wrapping the EBI human-annotated portion of SwissProt, a general database of gene protein products. Each of these agents, then, finds GO-annotated, close homologs of the unannotated gene and proposes the annotation of the homologs for the annotation of the new gene.

Task Agents. There are two new task agents, one is a domain-independent ontology agent using the FIPA ontology agent specification as a starting point. The ontology agent contains both the GO ontologies and several mappings from other symbologies (i.e. SwissProt terms developed before the advent of the GO ontologies) to GO terms. In fact, the Mouse IEA uses the Ontology agent to map some non-GO terms for certain records to GO terms. Although not indicated on the figure, some of the other organism DB IEA agents must map from GO ontology descriptive strings to the actual unique GO ID. The other service provided by the ontology agent (and not explicitly mentioned in the experimental FIPA Ontology Agent specification) is for the ontology reasoning agent to ask how to terms are related in an ontology. The Ontology Reasoning Agent uses this query to build a minimum spanning tree (in each of the three GO ontologies) between all the terms returned in all the homologies from all of the GO organism databases. This information can then be used to propose a likely annotation, and to display all of the information graphically for the biologist via the interface agent.

The Ontology Reasoning Agent implements an algorithm we have nicknamed GO-Del for deducing appropriate electronic GO annotations for unknown gene products. We can model the output of the organism information extraction agents as a set of BLAST "hits" $\langle$ gene-id, organism-database, e-value$\rangle$. The gene-id is a unique identifier of the homolog match within the organism-database with the corresponding expectation value or e-value (BLAST sequence similarity score). Each hit h has an set of annotation terms T_h, such that the hit h is annotated with every member of annotation set T_h. A member term t of the annotation set T_h is a tuple of the form $\langle$ go-id, go-term, go-evidence-code$\rangle$ where term t corresponds to the go-term identifed by the go-id identifier, and the go-evidence-code signals the means by the which the annotation term t was assigned to hit h. We will indicate H as the set of all hits returned from all IEAs, and T as the set of all terms in H.

Minimum Covering Graph Construction. The minimum covering graph (MCG) is a a spanning graph covering all the terms in set T, with the condition that all the ancestors of the terms in T are included. This covering graph is then minimized by pruning the path from the GO root (GO:0003673 or Gene_Ontology)

to the node with the greatest depth that covers all the terms in the set T, designating this node as the root of the MCG. By doing so, we eliminate all the ancestors of the MCG root which were the common ancestors to all of the terms in set T as well.

The minimum covering graph diffes from a minimum spanning graph in that when a node in the MCG has multiple parents, then all the paths to that node (i.e. via every parent) is added to the MSG. In the construction of a minimum spanning graph, a single path that minimized the total weight would have been chosen instead. We make this distinction to capture the multiple inheritance relationships embedded within the GO DAG.

Annotation Term Scoring. The MCG demarcates the boundary of annotation for the unknown gene or gene product. Thus, only the nodes present in the MCG will be used to annotate the unknown object. The relative scoring of these terms (or nodes) is done on the basis of the hits that were used to produce the MCG. Therefore, before we can score the nodes within the MCG, the quality of the hits will need to be evaluated.

Hit Score. Our algorithm allows for a hit to be scored along unlimited determinants. For the initial analysis, we identified two axes along which a hit might be differentiated, and thus scored: e-value score and GO evidence code[5].

The formula to calculate a hit score for a hit h is:

$$S(h) = w_{evalue} f_{evalue}(h) + w_{evidence} f_{evidence}(h)$$

where $w_{evalue} + w_{evidence} = 1$ provide relative weights for the two factors.

Hits within the GO databases with e-values close or equal to 0, are very similar to the sequence of the unknown object. As the e-value increases, the degree of similarity decreases. To represent this relationship, the evalue factor is calculated from the function

$$f_{evalue}(h) = (1/100)^{e-value(h)}$$

Thus, f_{evalue} is set to 1.0 for an exact match (e-value of zero). As the e-value increases, f_{evalue} drops off exponentially, and at an e-value of 1, f_{evalue} is reduced to 0.01.

The GO evidence codes indicate the method by which an annotation was assigned. Since this is a qualitative measure of reliability, a conversion table was created with the aid of a human expert to quantity the relative evidence factor score for each evidence code. The scoring was patterned on the "loose hierarchy" suggested by the GO Consortium[6]. For example, a Traceable Author Statement (TAS) ($f_{evidence} = 1.0$) contributes ten times as much to the hit score than an Inferred by Electronic annotation (IEA) code would ($f_{evidence} = 0.1$). When a hit has more than one annotation term associated with it, the best evidence factor is chosen.

Term Score. The term scores are calculated from the hit scores. A term can aggregate a score on the basis of the hit ratio of the hits annotated by that term. The hit ratio of a term node t (denoted as $R(t)$) is defined as the ratio of the sum of the hits at that node to the sum of all the hit scores in the MCG:

$$R(t) = \frac{\sum_{h \in H_t} S(h)}{\sum_{h \in H} S(h)}$$

where H_t is the subset of hits in H annotated with the term t.

[5]Later work may add phylogeny.

[6]`http://www.geneontology.org/GO.evidence.html`

Since the GO DAG represents subsumption class relationships indicated by edges representing is-a relationships, the support score for a child term should accrues to its parent terms. Thus, the term score can be recursively defined as

$$S_{term}(t) = \sum_{c \in children(t)} \frac{S_{term}(c)}{|parents(c)|} + R(t)$$

Assignment of Annotation Term(s). Selecting a final annotation term or terms is now straightforward as every term in the MCG is associated with a term score. Given a term score threshold, we traverse down from the root of the MCG until we arrive at a node whose term score is above or equal to the threshold, but all of its children are not. This node is then associated with the unknown object, for the given term score threshold. When more than one child has a term score greater than the threshold, both branches are taken and the process is recursed. A simple check is also required before a final annotation is chosen to ensure that no descendents of the term to be chosen has an equivalent term score - a possibility engendered by the score splitting required of terms with multiple parents.

Interface Agents. There are two possible functional annotation interface agents/applets. The first, ManGO (MANual GO), supports manual annotation of a local knowledgebase. Genes indicated by the LKBMA as unannotated, or annotated only by GODel (with the IEA Inferred by Electronic Annotation code) can be brought up, and the MCG dispayed for each on the ontologies. Frames allow editing of the information needed for a complete manual annotation and submission to the GO Consortium.

The second interface, GO-Figure![7], allows users anywhere on the internet to BLAST their own nucleotide or protein sequence against the functional annotation subsystem databases that have been annotated with GO terms. The requestor also provides an e-mail address. The search will return an email message with a URL link to three graphs depicting the GO annotations of similar sequences in the Molecular Function, Biological Process, and Cellular Component ontologies.

In the resulting graphs, colored boxes indicate term hits, with darker color indicating lower E-value. Box shape indicates IEA vs. non-IEA evidence codes. Colored boxes can be clicked on, and will bring up a table of all individual database hits and E-values. Links in this table are functional, and connect back to the original databases. Figure 6 shows an example graph produced for the Biological Process ontology, given an IL-1 beta protein as the input.

4.3 EST Processing

One way to broaden the applicability of the system is to accept more kinds of basic input data to the annotation process. For example, we could broaden the reach of the system by starting with ESTs (Expressed Sequence Tags) instead of complete sequences. Agents could wrap the standard software for creating sequences from this data, at which point the existing system can be used. The use of ESTs is part of a relatively inexpensive approach to sequencing where instead of directly sequencing genomic DNA, we instead use a method that produces many short sequences that partially overlap. By finding the overlaps in the short sequences, we can eventually reconstruct the entire sequence of each expressed gene. Essentially, this is a "shotgun" approach that relies on statistics and the sheer number of experiments to eventually produce complete sequences. Figure 7 shows a multi-agent subsystem for automating the processing of ESTs to produce consensus gene sequences.

As a side effect of this processing, information is produced that can be used to find Single Nucleotide Polymorphisms (SNPs). SNPs indicate a change of one nucleotide (A,T,C,G) in a single gene between

[7] http://udgenome.ags.udel.edu/gofigure

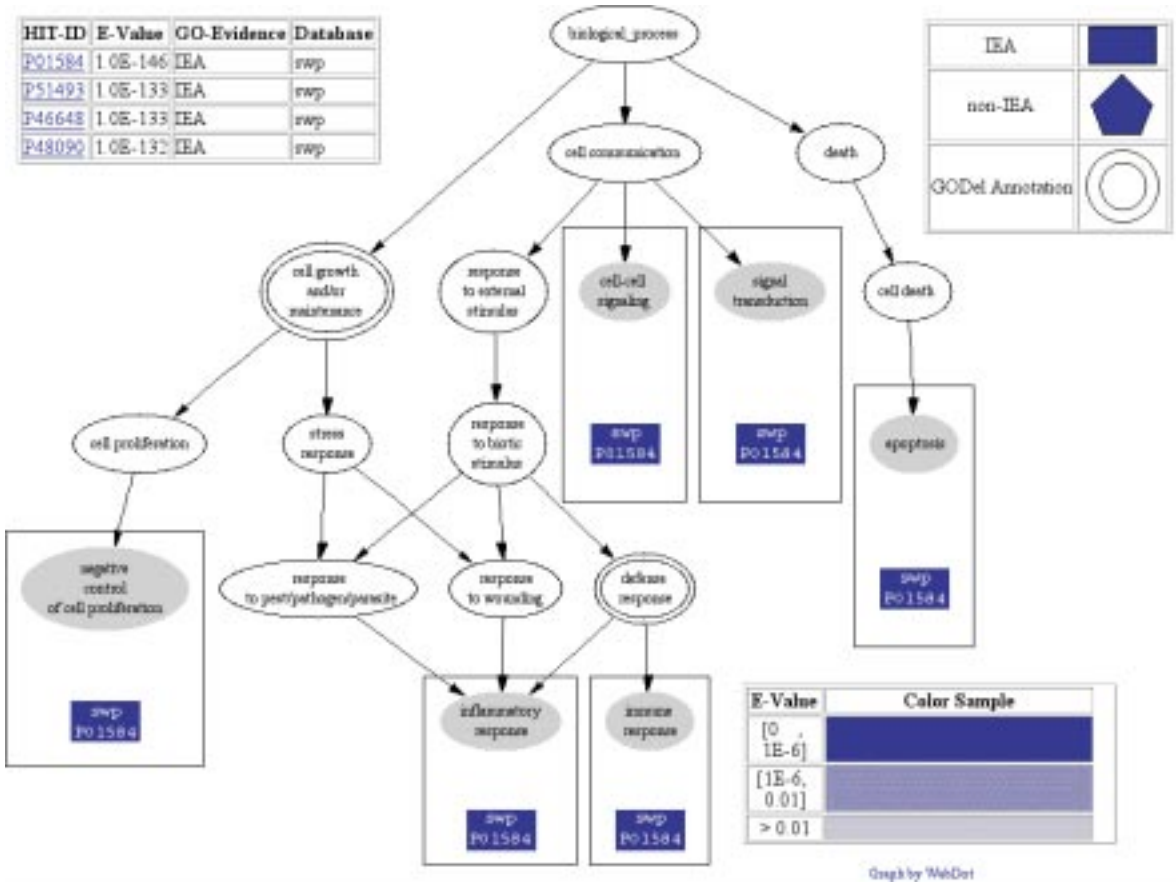

Figure 6: Go-Figure! output for IL-1 beta. Note that "IEA" here refers to Inferred by Electronic Annotation, and not Information Extraction Agent.

different individuals (often, conserved across strains or subspecies). These markers are very important for identification even if they do not have functional effects.

Information Extraction Agents. The process of consensus sequence building and SNP identification does not require any external information, so the only IEAs are the LKBMAs. Up until now, there has only been one LKBMA, responsible for the gene sequences and annotations. EST processing adds a second LKBMA responsible for storing the ESTS themselves and the associated information discussed below. Primarily, this is because (especially early on in a sequencing project) there will be thousands of ESTs that do not overlap to form contiguous sequences, and that ESTs may be added and processed almost daily.

Task Agents. There are three new domain-level task agents. The first deals with processing chromatographs. Essentially the chromatograph is a set of signals that indicate the relative strengths of the wavelengths associated with each luminous nucleotide tag. Several standard Unix analysis programs exist to process this data, essentially "calling" the best nucleotide for each position. The chromatograph processing agent wraps three analysis programs: Phred, which "calls" the chromatograph and also separately produces an uncertainty score for each nucleotide in the sequence; phd2fasta which converts this output into a standard (FASTA) format; and x-match which removes a part of the sequence that is a byproduct of the se-

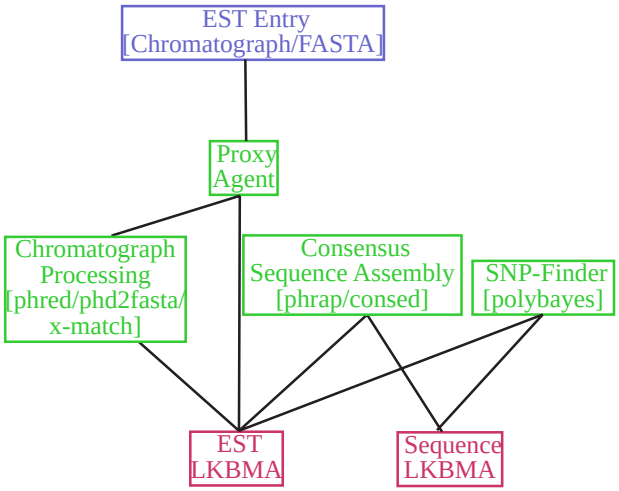

Figure 7: EST Processing Agent Organization

quencing method, and not actually part of the organism sequence. The consensus sequence assembly agent uses two more programs (Phrap and consed) on all the ESTs found so far to produce a set of candidate genes by appropriately splicing together the short EST sequences. This produces a set of candidate genes that can then be added to the gene sequence LKBMA and from which the various annotation processes described earlier may commence. Finally, a SNP-finder agent operates the PolyBayes program which uses the EST and Sequence KBs and the uncertainty scores produced by Phred to nominate possible single nucleotide polymorphisms. Each of the wrapped programs (especially Phred, Phrap, and PolyBayes) has a large number of parameters to control. Currently we set these in consultation with our biologist parteners, but as we get more experience we plan to experiment with various automated learning mechanisms for parameter adjustment.

Interface Agents. There is only one simple interface agent, to allow participants to enter data in the system. Preferably, this is chromatograph data from the sequencers, because the original chromatograph allows Phred to calculate the uncertainty associated with each nucleotide call. However, FASTA-format (simple "ATCG…" named strings) ESTs called from the original chromatographs can be accommodated. These can be used to build consensus sequences, but not for finding SNPs.

5 Gene Expression Processing

A new kind of genomic data is now being produced, that may swamp even the amount of sequencing data. This is so-called *gene expression* data, and indicates quantitatively how much a gene product is expressed in some location, under some conditions, at some point in time. We are developing an multi-agent system that uses available on-line genomic and metabolic pathway knowledge to extend gene expression analysis. By incorporating known relationships between genes, knowledge-based analysis of experimental expression data is significantly improved over purely statistical methods. Although this system has not yet been integrated into the existing agent community, eventually relevant genomic information will be made available to the system through the existing GenBank and SwissProt IEAs. Metabolic pathways of interest to the investigator are identified through a KEGG (Kyoto Encyclopedia of Genes and Genomes) database wrapper. Analysis of the gene expression data is performed through an agent that executes SAS, a statistical package

that includes clustering and PCA analysis methods. Results are to be presented to the user through web pages hyperlinked to relevant database entries.

Gene expression studies, which identify the set of active genes within a particular state, are beginning to explore the dynamic nature of intracellular behavior [14]. As cells respond to environmental or physiological changes of state, individual genes are induced and repressed and the corresponding messenger RNA (mRNA) and protein levels are modified, all according to a complicated but predetermined reaction network. However, interpreting these complex underlying networks from gene expression experiments is a difficult process. Current techniques for gene expression analysis have primarily focused on the use of clustering algorithms, which group genes of similar expression patterns together [15]. Based on the results of such analysis, researchers have identified conserved control regions (promoters) upstream of co-expressed genes that appear to be responsible for the similar expression behavior [7]. However, experimental gene expression data can be very noisy and the complicated pathways within organisms can generate coincidental expression patterns, which can significantly limit the benefits of standard cluster analysis. In order to separate gene co-regulation patterns from co-expression, the gene expression processing organization was developed to gather available pathway-level information in order to presort the expression data into functional categories. Thus, clustering of the reduced data set is much more likely to find genes that are actually regulated together. The system also promises to be useful in discovering regulatory connections between different pathways.

A diagram outlining the agents within the system is shown in Figure 8. Although the system currently only uses the KEGG database to identify pertinent metabolic pathways, other sites are available and shall be incorporated in the future. One advantage of using the KEGG database is that its gene/enzyme entries are organized by the EC (Enzyme Commission) ontology, and so are easily mapped to gene names specific to the organism of interest. The gene expression database agent for this system currently queries a local copy of the publicly available S. cerevisiae (yeast) whole cell expression data and allows the user to access data related to diauxic shift, cell cycle, and sporulation. Once the experimental data has been reduced by the system to only those genes within the metabolic pathways of interest, the SAS statistical package is then used to cluster the remaining data to identify what are hopefully closely regulated genes within the organism. The co-regulation of genes is confirmed though the identification of conserved promoter motifs within each gene's genetic code. Final results of an analysis run will be presented to the user in graphic form showing clustered expression patterns along with hyperlinks to relevant on-line database entries for further exploration.

Currently, we have automated access to the local database, the cluster analysis, and presentation to demonstrate the biological utility of the approach. Integration with the rest of the systems described here is planned.

6 Related Work

There has been significant work on general algorithms for query planning, selective materialization, and the optimization of these from the AI perspective, for example TSIMMIS [5], Information Manifold [23], Infosleuth [27], HERMES [1], SIMS [2], etc., and of course on applying agents as the way to embody these algorithms [24, 32, 12, 22].

In Biology, compared to the work being done to create the raw data, all the work on how to organize and retrieve it is relatively small. Most of the work in computer science directed to biological data has been in the area of heterogeneous databases, focusing on the semi-structured nature of much of the data that makes it very difficult to store usefully in commercial relational databases [6]. Some work has begun in applying the work on wrappers and mediators to biological databases, for example TAMBIS [31]. These systems

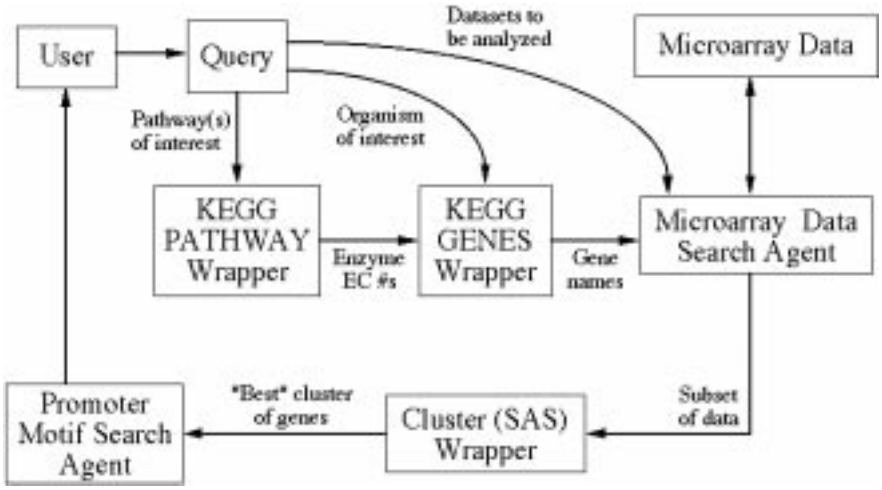

Figure 8: Overview of gene expression processing organization

differ from ours in that they are pure implementations of wrapper/mediator technology that are centralized, do not allow for dynamic changes in sources, support persistent queries, or consider secondary user utility in the form of time or other resource limitations.

Agent technology has been making some inroads in the area. The word "agent" with the popular connotation of a single computer program to do a user's bidding is found in the promotional material for Doubletwist (`www.doubletwist.com`). Here, an "agent" stands for a persistent query (e.g. "tell me if a new homolog is found in your database for the following sequence"). There is no collaboration or communication between agents.

We know of a few truly multi-agent projects in this domain. First, InfoSleuth has been used to annotate livestock genetic samples [13]. The flow of information is very similar to our system. However, the system is not set up for noticing changes in the public databases, for integrating new data sources on the fly, or for consideration of secondary user utility. Second, the EDITtoTrEMBL system [25] is another automated annotation system, based on the wrapper and mediator concept, for annotating proteins awaiting manual annotation and entry to SwissProt. Dispatcher agents control the application of potentially complex sequences of wrappers. Most importantly, this system supports the detection and possible revision of inconsistencies revealed between different annotations. Third, the GeneWeaver project [4] is another true multi-agent system for annotation of genomes. GeneWeaver has as a primary design criterion the observation that the source data is always changing, and so annotations need to be constantly updated. They also express the idea that new sources or analysis tools should be easy to integrate into the system, which plays to the open systems requirement, although they do not describe details. The primary differences are the way in which an open system is achieved (it is not clear that they use agent-level matchmaking, but rather possibly CORBA specifications) and that GeneWeaver is not based on a shared architecture that supports reasoning about secondary user utility. In comparison to the DECAF implementation, GeneWeaver uses CORBA/RMI rather than TCP/IP communication, and a simplified KQML-like language called BAL.

7 Discussion

The system described here is operational and normally available on the web at `http://udgenome.ags.udel.edu/herpes/`. This is a working prototype, and so the interface is strongly oriented to biologists only. In general, computational support for the *processes* that biologists use in analyzing data is primitive (Perl scripts) or non-existent. In less than 10 min, we were able to annotate the HVT-1 sequence, as well as store it in a queryable and web-publishable form. This impressed the biologists we work with, compared to manual annotation and flat ASCII files. Furthermore, we have recently added approximately 15 other publicly available herpesvirus sequences (e.g. several strains of Human herpesvirus, African swine fever virus, etc.). The resulting knowledgebase almost immediately resulted in queries by our local biologists that indicated possible interesting relationships that may result in future biological work. We are currently looking for feedback from viral biologists at other universities.

Other things about the system which have excited our biologist co-workers are the relative ease by which we can add new types of annotation or analysis information, and the fact that the system can be used to build similar systems for other organisms, such as the chicken. For example, the use of open system concepts such as a matchmaker allow the annotation agent to access and use new annotation services that were not available when it was initially written. Secondary user utility will become useful for the biologist when faced with making a simple office query vs. checking results before publication.

The underlying DECAF system has been evaluated in several ways, especially with respect to the use of parallel computational resources by a single agent (all of the DECAF components and all of the executable actions are run in parallel threads), and the efficacy of the DRU scheduler which efficiently solves a restricted subset of the design-to-criteria scheduling problem [18]. Running the gene annotation system as a truly multi-agent system results in true speedups, although most of the time is currently spent in remote database access (see Table 1). Parallel hardware for each agent will be useful for some of the more locally computationally intensive tasks involving EST processing.

BLASTP	PSort	Motif	Distr. BLASTP	Distr. PSort	Distrib. Motif
1994	1296	1833	775	346	809

Table 1: Average processing times on uniprocessor or distributed hardware

8 Conclusions and Future Work

In this paper we have discussed the very real problem of making some use of the tremendous amounts of genetic sequence information that are being produced. While there is much information publicly available over the web, accessing such information is different for each source and the results can only be used by a single researcher. Furthermore, the contents of these primary sources are changing all the time, and new sources and techniques for analysis are constantly being developed.

We cast this sequence annotation problem as a general information gathering problem, and proposed the use of multi-agent systems for implementation. Beyond the basic heterogeneous database problem that this problem represents, an MAS solution gives us mechanisms for dealing with changing data, the dynamic appearance of new sources, minding secondary utility characteristics for users, and of course the obvious distributed processing achievements of parallel development, concurrent processing, and the possibility for handling certain security or other organizational concerns (where part of the agent organization can mirror the human organization).

We currently are offering the system publicly on the web, with the known herpesvirus sequences. A second system based on chicken ESTs should be available by May of 2002. The GO-Figure! system is also available on the web seperately. We intend to broaden the annotation coverage and add more complex analyses. An example would be the estimation of the physical location of the gene as well as its function. Because biologists have long recorded certain QTLs (Quantitative Trait Loci) that indicate that a certain *physical region* is responsible for a trait (such as chickens with resistance to a certain disease), being able to see what genes are physically located in the QTL region is a strong indicator as to their high-level genetic function.

In general, we have not yet designed an interface that allows biologists to take full advantage of the materialized data —they are uncomfortable with complex query languages. We believe that it may be possible to build a graphical interface to allow a biologist, after some training, to create a commonly needed analysis query and to then save this for use in the future by that scientist, or others sharing the agent namespace.

Once we have ported the BioMAS to two organisms, we intend to begin addressing the problems of n organism knowledgebases. In a sense, we will begin to explore the questions involving the production of "peer-to-peer" knowledgebases using multi-agent systems techniques.

Finally, the next major subsystem will be agents to link and analyze gene expression data (which will in turn interoperate with the metabolic pathway analysis systems described above). This data needs to be linked with sequence and function data, to allow more powerful analysis. For example, linked to QTL data, this allows us to ask questions such as "what chemicals might prevent club root disease in cabbage?".

References

[1] S. Adali and V.S. Subrahmanian. Amalgamating knowledge bases, III: Distributed mediators. *International Journal of Intelligent Cooperative Information Systems*, 1994.

[2] Y. Arens and C.A. Knoblock. Intelligent caching: Selecting, representing, and reusing data in an information server. In *Proc. 3rd Intl. Conf. on Information and Knowledge Management*, 1994.

[3] D.A. Benson and et al. Genbank. *Nucleic Acids Res.*, 28:15–18, 2000. `http://www.ncbi.nlm.nih.gov`.

[4] K. Bryson, M. Luck, M. Joy, and D.T. Jones. Applying agents to bioinformatics in geneweaver. In *Proceedings of the Fourth International Workshop on Collaborative Information Agents*, 2000.

[5] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: integration of heterogeneous information sources. In *Proceedings of the Tenth Anniversary Meeting of the Information Processing Society of Japan*, December 1994.

[6] S. B. Davidson and et al. Biokleisli:a digital library for biomedical researchers. *Intnl. J. on Digital Libraries*, 1(1):36–53, 1997.

[7] K. Decker, X. Zheng, and C. Schmidt. A multi-agent system for automated genomic annotation. In *Proceedings of the 5th Intl. Conf. on Autonomous Agents*, Montreal, 2001.

[8] K. S. Decker, A. Pannu, K. Sycara, and M. Williamson. Designing behaviors for information agents. In *Proceedings of the 1st Intl. Conf. on Autonomous Agents*, pages 404–413, Marina del Rey, February 1997.

[9] K. S. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 578–583, Nagoya, Japan, August 1997.

[10] Keith S. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD thesis, University of Massachusetts, 1995. `http://dis.cs.umass.edu/~decker/thesis.html`.

[11] Keith S. Decker and Victor R. Lesser. Quantitative modeling of complex computational task environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 217–224, Washington, July 1993.

[12] Keith S. Decker and Katia Sycara. Intelligent adaptive information agents. *Journal of Intelligent Information Systems*, 9(3):239–260, 1997.

[13] L. Deschaine, R. Brice, and M. Nodine. Use of infosleuth to coordinate information acquisition, tracking, and analysis in complex applications. Technical Report MCC-INSL–008-00, MCC, 2000.

[14] J. L. DiRisi, V.R. Iyer, and P.O. Brown. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, 278:680–686, 1997.

[15] M.B. Eisen, P.T. Spellman, P.O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc. Nat. Acad. Sci.*

[16] K. Erol, D. Nau, and J. Hendler. Semantics for hierarchical task-network planning. Technical report CS-TR-3239, UMIACS-TR-94-31, Computer Science Dept., University of Maryland, 1994.

[17] J. Graham and K.S. Decker. Towards a distributed, environment-centered agent framework. In N.R. Jennings and Y. Lesperance, editors, *Intelligent Agents VI*, LNAI-1757, pages 290–304. Springer Verlag, 2000.

[18] John Graham. *Real-time Scheduling in Multi-agent Systems*. PhD thesis, University of Delaware, 2001.

[19] T. Harvey, K. Decker, and O. Rambow. Integrating the communicative plans of multiple, independent agents. In *Workshop on Communicative Agents: The use of natural language in embodied systems*, 1999. Autonomous Agents 99.

[20] J. Hendler and Merwyn Taylor Kilian Stoffel. Advances in high performance knowledge representation. Technical Report CS-TR-3672, University of Maryland Institute for Advanced Computer Studies, 1996. Also cross-referenced as UMIACS-TR-96-56.

[21] B. Horling, V. Lesser, R. Vincent, A. Bazzan, and P. Xuan. Diagnosis as an integral part of multi-agent adaptability. Tech Report CS-TR-99-03, UMass, 1999.

[22] L. Kerschberg. Knowledge rovers: cooperative intelligent agent support for enterprise information architectures,. In P. Kandzia and M. Klusch, editors, *Cooperative Information Agents*, LNAI-1202. Springer-Verlag, 1997.

[23] T. Kirk, A. Levy, J. Sagiv, and D. Srivastav. The information manifold. Technical report, AT&T Bell Labs, 1995.

[24] C.A. Knoblock, Y. Arens, and C. Hsu. Cooperating agents for information retrieval. In *Proc. 2nd Intl. Conf. on Cooperative Information Systems*. Univ. of Toronto Press, 1994.

[25] Steffen Möller and Michael Schroeder. Consistent integration of non-reliable heterogeneous information applied to the annotation of transmembrane proteins. *Journal of Computing and Chemistry*, 2001. To appear.

[26] I. Muslea, S. Minton, and C. Knobloch. Stalker: Learning expectation rules for simistructured web-based information sources. In *Papers from the 1998 Workshop on AI and Information Gathering*, 1998. also Technical Report ws-98-14, University of Southern California.

[27] M. Nodine and A. Unruh. Facilitating open communication in agent systems: the infosleuth infrastructure. In M. Singh, A. Rao, and M. Wooldridge, editors, *Intelligent Agents IV*, pages 281–295. Springer-Verlag, 1998.

[28] A.S. Rao and M.P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 312–319, San Francisco, June 1995. AAAI Press.

[29] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, Cambridge, Mass., 1994.

[30] R. Simmons. Structured control for autonomous robots. *IEEE Trans. on Robotics and Automation*, 10(1), February 1994.

[31] R. Stevens and et al. Tambis: Transparent access to multiple bioinformatics information sources. *Bioinformatics*, 16(2):184–185, 2000.

[32] K. Sycara, K. S. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert*, 11(6):36–46, December 1996.

[33] The Gene Ontology Consortium. Gene ontolgy: tool for the unification of biology. *Nature Genetics*, 25(1):25–29, May 2000.

[34] T. Wagner, A. Garvey, and V. Lesser. Complex goal criteria and its application in design-to-criteria scheduling. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, July 1997.

[35] M.P. Wellman and J. Doyle. Modular utility representation for decision-theoretic planning. In *Proc. fo the First Intl. Conf. on Artificial Intelligence Planning Systems*, pages 236–242, June 1992.

[36] M. Williamson, K. S. Decker, and K. Sycara. Executing decision-theoretic plans in multi-agent environments. In *AAAI Fall Symposium on Plan Execution*, November 1996. AAAI Report FS-96-01.

[37] M. Williamson, K. S. Decker, and K. Sycara. Unified information and control flow in hierarchical task networks. In *Proceedings of the AAAI-96 workshop on Theories of Planning, Action, and Control*, 1996.

[38] M. Wooldridge and N.R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.