A Multi-Agent System for Automated Genomic Annotation

Keith Decker
Dept. of Computer Science
University of Delaware
Newark, DE 19716
decker@cis.udel.edu

Xiaojing Zheng
Dept. of Computer Science
University of Delaware
Newark, DE 19716
zheng@cis.udel.edu

Carl Schmidt
Dept. of Animal & Food Science
University of Delaware
Newark, DE 19716
schmidt@udel.edu

ABSTRACT

Massive amounts of raw data are currently being generated by biologists while sequencing organisms. Outside of the largest, high-profile projects such as the Human Genome Project, most of this raw data must be analyzed through the piecemeal application of various computer programs and searches of various public web databases. Due to the inexperience and lack of training, both the raw data and any valuable derived knowledge will remain generally unavailable except in published textual forms.

Multi-agent information gathering systems have a lot to contribute to these efforts, even at the current state of the art. We have used DECAF, a multi-agent system toolkit based on RETSINA and TAEMS, to construct a prototype multi-agent system for automated annotation and database storage of sequencing data for herpesviruses. The resulting system eliminates tedious and always out-of-date hand analyses, makes the data and annotations available for other researchers (or agent systems), and provides a level of query processing beyond even some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.

Keywords

information agents, lessons learned from deployed agents

1. INTRODUCTION

The rapidly accelerating rate at which organisms are being genomically sequenced has changed the practice of biology. Most important has been the realization that a tremendous amount of genetic material is similar from organism to organism, even when they are as outwardly different as a budding yeast, fruit fly, or human being. This means that if biologists studying the yeast can figure out what a certain gene does—its *function*—that others 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.

AGENTS'01, May 28-June 1, 2001, Montréal, Quebec, Canada.

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 the gene product does its work [24]. For example, biologists can get clues to gene function by:

- looking for certain patterns of amino acids (called *motifs*) 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
- 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. In this paper, we will focus on the *information gathering* problem, one for which appropriate multi-agent systems tools have already been developed. What we have is a large set of potentially large, 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 hundreds to hundreds of thousands of possible genes, for which functions must be hypothesized and which will lead to experiments for confirming gene function, looking for patterns or similarities, and generally focusing their priorities. For the case of all but the largest and well-funded sequencing projects, this must be done by hand by a single researcher.

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

This paper will describe a prototype multi-agent system built using the DECAF agent toolkit for the automated annotation of herpesvirus sequences, including the newly sequenced herpesvirus of turkey[1], HVT-1. First we will describe what we mean by information gathering, then describe the DECAF agent architecture and toolkit support for information gathering, then describe the herpesvirus annotation problem as an information gathering problem, and finally discuss the organization of the resulting system. This system is now sucessfully in use by biologists at the University of Delaware, replacing the previous system of laborious hand-annotation, and will be available publically to other biologists over the web. We conclude with a review of related work and some of the computer science-oriented problems that we have not yet addressed.

2. 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)[9, 10], military strategic information (integration of friendly troop movements, enemy observations, weather, satellite data, civilian communications)[14], 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[2] for building information gathering systems. The RETSINA approach consists of three general classes of agents [23, 10]:

- 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 [9], 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
- 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

[1] That's the bird, not the country.

[2] 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.

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 [13]. 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, execution, and coordination, 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 [10, 23]: communication, planning, scheduling, execution monitoring, and coordination. 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.

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[21]) 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

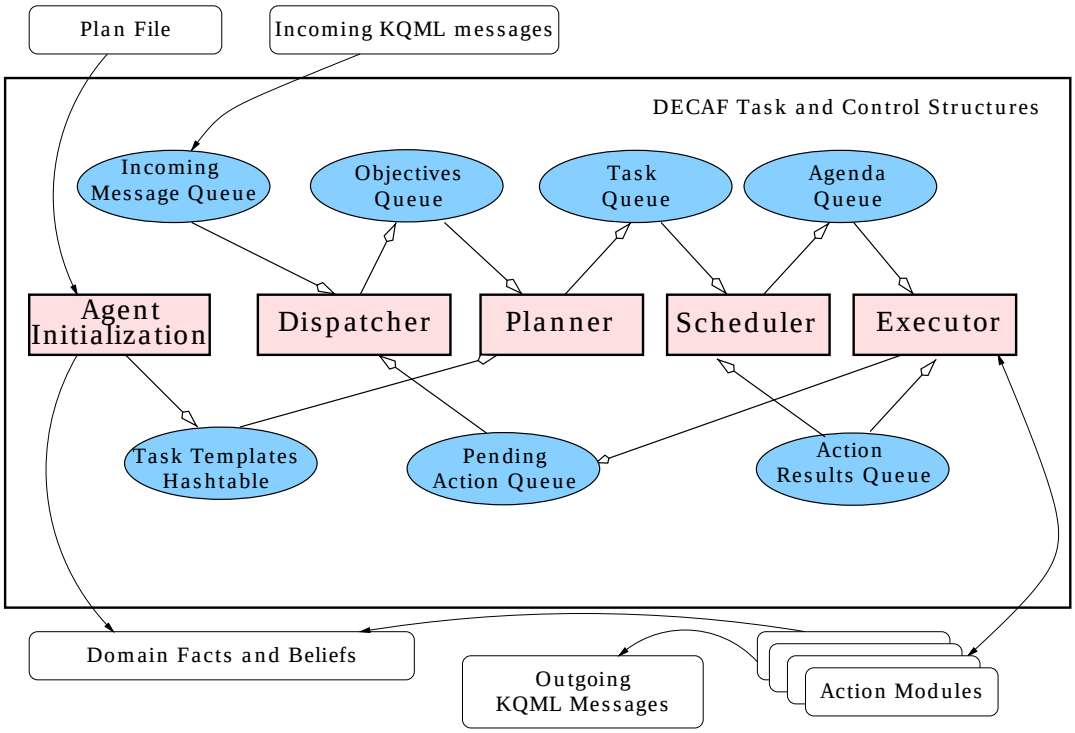

Figure 1: DECAF Architecture Overview

allow the scheduler to select an "optimal" path for task completion.[3]

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.

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 [27, 26], 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 [8, 25].

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

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 [26]. 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.

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[4]). 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" [28] 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.

3.3 DECAF Support for Info Gathering

How can DECAF support the construction, maintenance, and performance of information gathering systems? DECAF provides core architectural support for one feature of the information gathering problem, that of 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 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 persistent 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. The **Agent Name Server (ANS)** ("white pages") is an essential component for agent communication. It works in a fashion similar to DNS (Domain Name Service) by resolving agent names to host and port addresses. The **Matchmaker** serves as a "yellow pages" to assist agents in finding services needed for task completion. The **Broker** agent acts as a kind of "middle manager" to assist an agent with collections of services. The broker can now provide a larger service than any single provider can, and often manage a large group of agents more effectively [11]. A **Proxy** agent allows web page Java applets to communicate with DECAF agents that are not located on the same server as the applet. The **Agent Management Agent (AMA)** allows MAS designers a look at the entire running set of agents spread out across the internet that share a single agent name server. This allows designers to query the status of individual agents and watch or record message passing traffic.

3.3.1 Information Extraction Agent Shell

The main functions of an information extraction agent (IEA) are [9]: 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 [19], 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 [15] 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.

4. A DECAF MULTI-AGENT SYSTEM FOR HERPESVIRUS ANNOTATION

These tools can be put to use to create a prototype multi-agent system for annotating herpesvirus sequences. As a prototype, we have chosen to simplify several features that would be in a full system. First, all annotations are materialized at the local sequence database. Secondly, we treat some data as strings which really contain more information, in particular gene function (we'll explain this more in Future Work, but suffice to say, this is the way it is treated in almost all existing databases). With that caveat, the organization of the system is shown in Figure 2.

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

[4]The full set of quality accumulation functions, including alternate definitions for AND and OR, is discussed in [7].

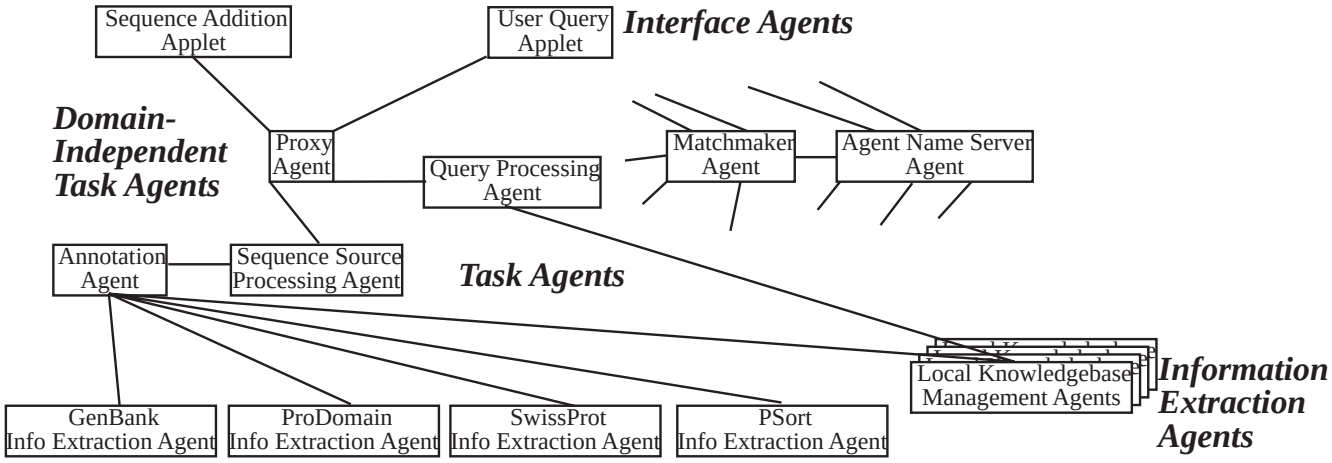

Figure 2: DECAF Multi-Agent System for Genomic Annotation

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. The ProDomain wrapper allows access to other information about the encoded protein.

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 shareable annotated sequence databases using the existing wrappers and other analysis tools as they are developed, without having to necessarily download and install them themselves. Using a PARKA-DB knowledgebase allows efficient, modern relational data storage and query as well as limited inferencing [15].

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. The main function of this agent is to test this input for internal consistency, and eventually to aid in ontologically correct functional annotation (see future work).

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 public query interface to the herpesvirus KB is currently available as indicated in the appendix to this paper. 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.

5. 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 [17], Infosleuth [20], HERMES [1], SIMS [2], etc., and of course on applying agents as the way to embody these algorithms [18, 23, 10, 16].

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 [22]. These systems 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[5]. 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").

[5]www.doubletwist.com

There is no collaboration or communication between agents.

We know of two truly multi-agent projects in this domain. First, InfoSleuth has been used to annotate livestock genetic samples [12]. 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. Secondly, 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.

6. EVALUATION

The system described here is operational and available on the web at `http://udgenome.ags.udel.edu/herpes/`. This is a *real* 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-existant. 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 25 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. This summer we will begin testing with viral biologists from 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 (see Future Work). 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. We have not yet evaluated the usefulness of reasoning about secondary user utility from the standpoint of the biologist, but it does provide the agent programmer ways of building robust systems with alternative methods to achieve goals.

7. 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 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.

We currently are offering the system publicly on the web, and are populating it with the known herpesvirus sequences. Now that the core functionality is complete, we would like to broaden the coverage of both annotation and add other, more complex analyses. 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 could be used. An example of a more complex analysis 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.

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. 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?".

Finally, the most difficult problem is that of annotating genetic function itself. Unfortunately, the millions of genes sequenced so far have fairly haphazard (from a computer scientist's perspective) functional annotation: simply textual 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 [24]. A main area of our future research is to both support the use of this ontology by biologists as sequences are added to the system, and to use it to allow even more powerful analysis of the resulting databases.

8. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grants IIS-9812764 and IIS-9733004.

9. REFERENCES

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

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

[8] K. S. Decker and V. 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.

[9] 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, Feb. 1997.

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

[11] 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, Aug. 1997.

[12] 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.

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

[14] 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.

[15] J. Hendler and M. T. 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.

[16] 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.

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

[18] C. 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.

[19] 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.

[20] 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.

[21] A. Rao and M. 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.

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

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

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

[25] 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.

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

[27] 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.

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