# Agent Interaction for Bioinformatics Data Management

K. Bryson[1], M. Luck[2], M. Joy[1], D.T. Jones[3]

1. Department of Computer Science, University of Warwick, CV4 7AL, UK
2. Electronics and Computer Science, University of Southampton, SO17 1BJ, UK
3. Department of Biological Sciences, Brunel University, UB8 3PH, UK

EMAIL: {bryson, M.S.Joy}@dcs.warwick.ac.uk, mml@ecs.soton.ac.uk, David.Jones@brunel.ac.uk

**Abstract**

As genome projects produce increasingly large quantities of sequence data, fast and reliable sequence analysis methods are required. Basic methods for comparing pairs of sequences or detecting patterns are well-developed, and now the key problem in analysing this genomic data is how to integrate the software and primary databases in a flexible and robust way. The wide range of available programs conform to very different input, output and processing requirements, typically with little consideration given to issues of integration. Key to addressing these issues appropriately is not to consider them as a result of the biological domain, but instead as an information processing problem that suggests nothing as much as an agent-based approach. In this paper, we introduce GeneWeaver, a multi-agent system for bioinformatics, and describe in detail the agent interactions which allow the integration and management of analysis methods and data. The system does not offer new methods but instead manages existing databases and analysis tools in an effective and flexible way, and facilitates easy and dynamic growth.

## 1 Introduction

The agent metaphor has been extremely successful in engaging wide interest and research, with some substantial success in terms of development of large agent systems. Apart from the obvious reasons of impact and intuition, the swell of interest in agents has typically been attributed to key changes and advances in the technological landscape over a number of years in recent times. Perhaps the most dramatic of these changes has been the emergence of the World-Wide Web (WWW), a double-edged sword which, on the one hand has opened up a wealth of resources in an accessible way and provided ready technologies for remote distribution of information that brings with it, on the other hand, a new set of problems relating to information gathering, for example [16, 20]. As far as agents are concerned, both the benefits and the difficulties that have arisen as a result of the Web are grist to the mill of agent research and development. The distribution of information and associated technologies lend themselves almost ideally to use by, in and for multi-agent systems. The dual aspect of this interaction with the WWW has thus been a major driving force.

However, the Web in itself is not the only factor, though its sudden and dramatic appearance, and its pervasive nature might mask other issues. In particular, advances in object technology, and more specifically distributed object technology, have provided an infrastructure without which the development of large-scale agent systems would become much more difficult and less effective and, without a doubt, agent techniques and technologies would become less transferable. For example,

the CORBA distributed computing platform [38] to handle low-level interoperation of heterogeneous distributed components, is a valuable piece of technology that can underpin the development of agent systems without the need for re-invention of fundamental techniques.

It has been argued elsewhere [29] that the success of agent systems is due to the timely coincidence of a maturity in some related fields and specific developments in others that have converged in a particular way, catalysed by the agent metaphor, to describe the current state of the art. With this maturing of the technology, and the increasing acceptance of agents and their deployment in commercial and industrial applications, agents can be regarded as moving out of the laboratory. The adoption of agent technology for use in fielded applications is an important milestone in the development of the field, and marks the start of the transition from prototypes and demonstrators to the commercial products that can provide further impetus.

In this paper, we describe a multi-agent system that involves all of these issues in its application to the very real and demanding problems of data integration and management for the life sciences. The vast quantities of data being rapidly generated by various sequencing efforts, the global distribution of available but remote databases that are continually updated, the existence of numerous analysis programs to be applied to sequence data in pursuit of determining protein structure and function, all point to the suitability of an agent-based approach.

We begin with a brief introduction to the problem domain, and explain how it leads to the current situation in which systems such as the one we describe here are vital. Then we introduce the prototype GeneWeaver agent community, a multi-agent system for managing the task of genome analysis, describing the agents involved, and the agent architecture. It should be noted that GeneWeaver can also be viewed as providing a more generic framework for a range of such agent-based systems. Finally, inter-agent communication in support of data and techniqe management is described, and some relevant data management protocols and scenarios are presented.

## 2 Genome Analysis and Protein Structure Prediction

The Human Genome Project [12] has been running since 1990 as a multi-billion dollar project, originally to find the estimated 100,000 or more human genes and to determine the sequence of the 3-billion DNA basepairs making up the human genome. A draft of the sequence has now been published [14] and it is estimated that we have between 20,000 and 30,000 genes. However, sequencing the human genome is only the beginning, and a substantial effort now needs to take place to organise and make sense of the data so that it may be used by life scientists in treating disease, for example. Moreover, the human genome is not the only genome being sequenced, with over one hundred other genomes having already been sequenced, and the flood of data constantly increasing, including organisms ranging in size from small viruses to mice. All the data needs to be analysed and integrated to provide a coherent and useful overall picture.

### 2.1 Flow of Biological Information

In this section, we begin by outlining the biology underlying the problem domain, but of necessity give only a brief treatment. The interested reader can refer to numerous introductory texts, such as [26].

Typically, an organism stores its hereditary material using DNA molecules, which are passed from one generation to the next. The data stored in these molecules can be simply represented as strings composed from four letters (G, C, A and T). Now, in a cell, a large number of short segments of the DNA molecule are *transcribed* into complementary strands of RNA, which essentially represent the

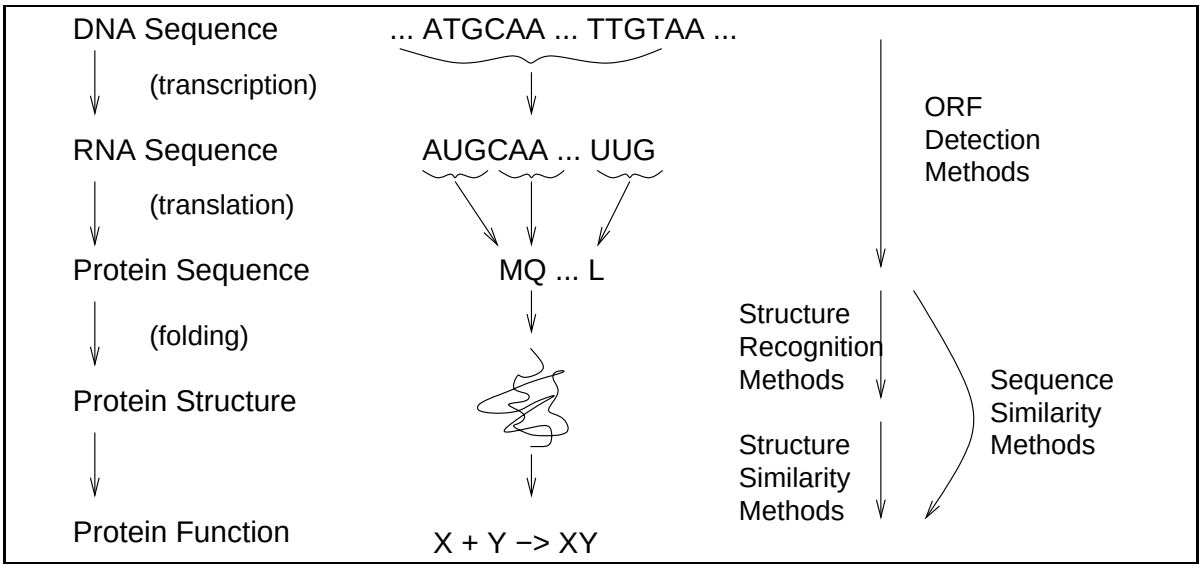


Figure 1: The flow of biological information within a cell and its computational processing.

*genes* of the organism. Similar to DNA, the data in each RNA molecule can be represented by a string composed from four letters (G, C, A and U) in which the T from the DNA is now replaced by U, as illustrated in Figure 1. Typically, the RNA molecules are then *translated* into protein molecules that can be represented as strings composed from 20 different letters. Each consecutive group of 3 RNA letters is translated into a single protein letter according to the genetic code for the organism. The protein molecule then *folds* into a precise three-dimensional structure, which usually contains a number of *active sites* on its surface where other proteins or small molecules may bind, and this gives the protein its unique biological function. For example, Figure 1 shows a resulting protein with the function of taking two molecules (X and Y) and combining them into a single molecule (XY). This would only be one small part of an enormous network of such reactions, which would result in the overall function of the particular type of cell.

While this very simplified view of the cellular processes involved offers some insight into the underlying biology, some more complicated stages (such as RNA processing) have been omitted completely. In addition, although our presentation has suggested that one process is universal, there are exceptions to the general rules, as with the HIV virus, which stores its genome using RNA instead of DNA, but we will not consider such issues further in this paper.

Figure 1 summarises the flow of biological information taking place in a cell as described above, together with the associated computational processing of this data, that we describe next. On the left hand side of the figure, aspects of different molecules involved in the production of a protein with a specific function are shown with a more concrete representation of the data involved, and its transformation, being given in the central part of the figure through an example. The right hand side shows some classes of techniques that a bioinformatician may use to elucidate the different types of information from the genomic DNA sequence of an organism. It includes the computational data processing, and shows that it may involve a number of different paths of data analysis.

## 2.2 Computational Processing

As indicated above, the rate at which this primary genetic data is being produced at present, is extremely rapid, and increasing. There is consequently a huge amount of genome data that is freely available across the Internet, typically stored in flat file databases providing DNA sequences. The problem of working out what each gene does, especially in light of the potential benefits of doing so, is therefore correspondingly pressing, and one which is meriting much attention from various scientific communities.

Traditionally, the first task for the bioinformatician is to use what are known as *ORF detection methods* on the DNA to determine the different protein sequences that are encoded in it. For each protein sequence, a variety of techniqes, such as *sequence similarity* and *motif detection*, may be used to try to elucidate the function of the protein. Similarly, a range of *structure recognition* techniques may be used to determine the three-dimensional structure of the protein which may then, in turn, help in determining its function.

This process of identifying genes and predicting the structure of the encoded proteins is thus fairly labour-intensive, made worse by requiring substantial expert knowledge concerning the available analysis techniques and the underlying data. However, the steps involved are all computer-based tasks: scanning sequence databases for similar sequences, collecting the matching sequences, constructing alignments of the sequences, and trying to infer the function of the sequence from annotations of the matched proteins (for which the function is already known). All of this requires use of the wide range of available tools for these tasks, which sometimes offer results that agree, but often do not (though they typically provide confidence scores that enable relatively easy interpretation).

Many tools are available to perform these tasks, but they are generally standalone programs that are not integrated with each other and require expert users to perform each stage manually and combine them in appropriate ways. For example, the process of trying to find a matching sequence might result in turning up an annotated gene, but the annotations include a lot of spurious information as well as the important functional information. The problem here is distilling this relevant information, which is not at all difficult for an expert, but which might prove problematic for a less experienced user. With the amount of data that is being generated, this kind of expertise is critical.

Finally, the information gained from this analysis is added back into relevant databases as a resource in its own right, with entries usually including large amounts of natural language in addition to the raw data. For example, a very small entry of the data held about a protein in the SWISSPROT database is shown in Figure 2. The updating of this data over time as new information emerges adds an additional dimension to the data management requirements. Analysis programs may generate or modify data that other programs also use, requiring the different sources of information and the different programs to be managed effectively to ensure that the systems as a whole is coherent, consistent and synchronised.

If we consider the kinds of problems raised by these issues, including filtering and prioritising information resulting from matched proteins, integrating several distinct analysis programs possibly in sophisticated ways, managing multiple remote sources of data in different formats, and so on, no solution for automation suggests itself quite as much as a multi-agent approach. In fact, this *kind* of problem is not really novel — it fits what might be considered a standard model of a multi-agent system in a traditional information systems domain with the addition of some extra complications, and a different set of data.

```
ID   HD_FUGRU       STANDARD;      PRT;  3148 AA.
AC   P51112;
DT   01-OCT-1996 (Rel. 34, Created)
DT   01-OCT-1996 (Rel. 34, Last sequence update)
DT   01-OCT-1996 (Rel. 34, Last annotation update)
DE   HUNTINGTIN (HUNTINGTON'S DISEASE PROTEIN HOMOLOG) (HD PROTEIN).
GN   HD.
OS   Fugu rubripes (Japanese pufferfish) (Takifugu rubripes).
OC   Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Actinopterygii;
OC   Neopterygii; Teleostei; Euteleostei; Acanthopterygii; Percomorpha;
OC   Tetraodontiformes; Tetraodontoidei; Tetraodontidae; Fugu.
RN   [1]
RP   SEQUENCE FROM N.A.
RX   MEDLINE; 95375788.
RA   BAXENDALE S., ABDULLA S., ELGAR G., BUCK D., BERKS M., MICKLEM G.,
RA   DURBIN R., BATES G., BRENNER S., BECK S., LEHRACH H.;
RT   "Comparative sequence analysis of the human and pufferfish
RT   Huntington's disease genes.";
RL   Nat. Genet. 10:67-76(1995).
CC   -!- FUNCTION: MAY PLAY A ROLE IN MICROTUBULE-MEDIATED TRANSPORT OR
CC       VESICLE FUNCTION.
CC   -!- SUBCELLULAR LOCATION: CYTOPLASMIC (BY SIMILARITY).
CC   -!- POLYMORPHISM: THE POLY-GLN REGION (FOUR RESIDUES) DOES NOT APPEAR
CC       TO BE POLYMORPHIC, EXPLAINING THE ABSENCE OF A HD-LIKE DISORDER.
CC   -!- SIMILARITY: CONTAINS 10 HEAT REPEATS.
CC   -!- SIMILARITY: BELONGS TO THE HUNGTINTIN FAMILY.
CC   ---------------------------------------------------------------------
CC   This SWISS-PROT entry is copyright. It is produced through a collaboration
CC   between  the Swiss Institute of Bioinformatics  and the  EMBL outstation -
CC   the European Bioinformatics Institute.  There are no  restrictions on  its
CC   use  by  non-profit  institutions as long  as its content  is  in  no  way
CC   modified and this statement is not removed.  Usage  by  and for commercial
CC   entities requires a license agreement (See http://www.isb-sib.ch/announce/
CC   or send an email to license@isb-sib.ch).
CC   ---------------------------------------------------------------------
DR   EMBL; X82939; CAA58112.1; -.
KW   Repeat.
FT   DOMAIN      148    272       HEAT REPEATS DOMAIN 1.
FT   DOMAIN      701    898       HEAT REPEATS DOMAIN 2.
FT   DOMAIN     1527   1568       HEAT REPEATS DOMAIN 3.
FT   DOMAIN       18     21       POLY-GLN.
FT   DOMAIN      679    682       POLY-ALA.
FT   DOMAIN     1104   1108       POLY-SER.
SQ   SEQUENCE   3148 AA;  348932 MW;  CBB3AA6A CRC32;
     MATMEKLMKA FESLKSFQQQ QGPPTAEEIV QRQKKEQATT KKDRVSHCLT ICENIVAQSL
     RTSPEFQKLL GIAMEMFLLC SDDSESDVRM VADECLNRII KALMDSNLPR LQLELYKEIK
     KNGASRSLRA ALWRFAELAH LIRPQKCRPY LVNLLPCLTR ITKRQEETIQ ETLAAAMPKI
.
.
.
     SLSCFFVSAS TSQWISALLP HVISRMGSSD VVDVNLFCLV AMDFYRHQID EELDRRAFQS
     VFETVASPGS PYFQLLACLQ SIHQDKSL
//
```

Figure 2: An example SWISSPROT database entry.

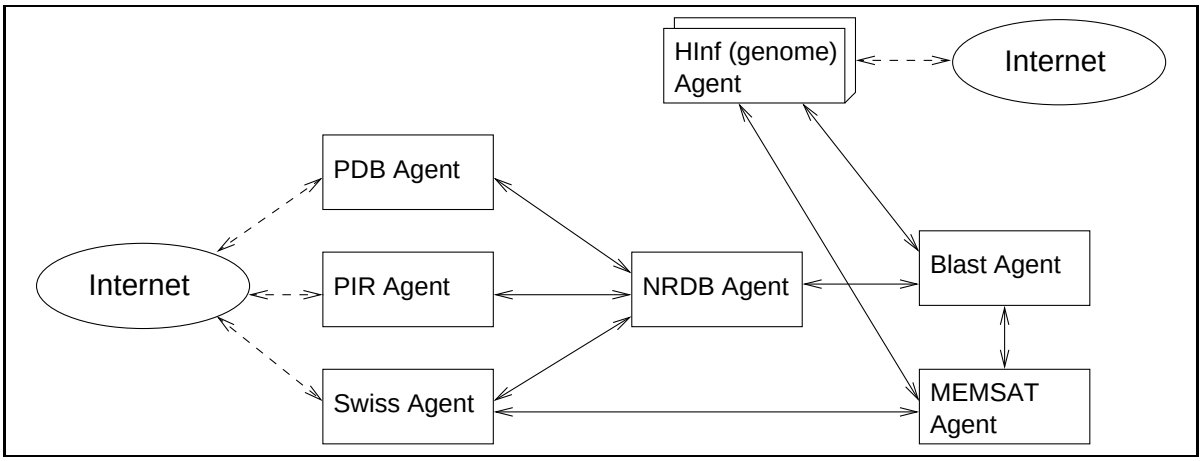


Figure 3: GeneWeaver agent community

# 3 GeneWeaver Agent Community

GeneWeaver is a multi-agent system aimed at addressing many of the problems concerning the management of data and analysis techniques in the domain of bioinformatics, as discussed until now. It comprises a community of agents that interact with each other, each performing some distinct task, in an effort to automate the processes involved in, for example, determining protein function. Agents in the system can be concerned with management of the primary databases, performing sequence analyses using existing tools, or with storing and presenting resulting information. The important point to note is that the system does not offer new methods for performing these tasks, but organises existing ones for the most effective and flexible operation. This section provides an overview of the system through the agents within it.

Figure 3 illustrates the overall perspective of GeneWeaver in that it contains the different classes of agents and shows how they inter-relate. At the left side, PDB agent, PIR agent and SWISS agent all manage the primary sequence databases indicated by their names (as described later), and interact with the NRDB (non-redundant database) agent, which combines their data. At the right edge of the figure, the calculation agents (including the BLAST agent and the MEMSAT agent that perform specific analysis tasks) attempt to annotate sequences in the genomes using relevant programs, again indicated by their names. Finally, at the top, a number of genome agents manage the genomes for particular organisms, and request analysis from the calculation agents. At each point of external interaction, agents typically receive and provide information via the Internet. In the figure, dashed lines indicate agents communicating to and from the Internet using http or ftp protocols. Solid lines represent the main communication pathways between the agents using an agent communication language. The broker agent (see below) is omitted for clarity.

There are five types of agent present in the GeneWeaver community.

- *Broker agents*, which are not shown in Figure 3 since they are *facilitators* rather than points of functionality, are needed to register information about other agents in the community.
- *Primary database agents* are needed to manage remote primary sequence databases, and keep the data contained in them up-to-date and in a format that allows other agents to query that data.
- *Non-redundant database agents* construct and maintain non-redundant databases from the data managed by other primary database agents in the community.

- *Calculation agents* encapsulate some pre-existing methods or tools for analysis of sequence data, and attempt to determine the structure or function of a sequence. Whenever possible, they are also responsible for constructing and managing any underlying data that they rely on.
- *Genome agents* are responsible for managing the genomic information for a particular organism.

Each of these agents is considered in more detail below.

### 3.1 Broker Agents

The potential size of the GeneWeaver agent community can be large, with the number of agents determined by the number of external databases used, the number of sequence comparison programs, and the extent of various other utility functions. Since each of these aspects must be wrapped up in an agent, with numerous interactions required between agents, brokers are needed to provide what is, in effect, a *yellow pages* service for all agents in the system. (Different forms of services offered might range from the simple yellow pages service to more sophisticated *matchmaking* [27, 28], but such issues are beyond the scope of this paper.)

A broker provides information about all the agents currently present in the community, including the communication methods they support. Each agent is required to register information about itself with the broker to be recognized within the community. Conversely, agents can query the broker for information about other agents in the community. They may also subscribe to the broker's information and will be notified when it changes as, for example, when agents join or leave the community.

### 3.2 Primary Database Agents

Although the principle of combining different methods and different information sources seems simple, in practice there are a number of difficulties that must be faced. Primary data sources (the primary sequence data banks and the structure data banks, for example) encode their information in different ways, not only in terms of the basic formatting, but also in the terminology used. For example, different keyword sets are used for each data bank so that understanding and interpreting the data (both of primary data sources and the results from analysis programs) is not a trivial task.

Primary databases are provided externally, under the control of third-parties who may change them at any time. In many cases, the data is largely unstructured, and is often available in the form of flat files. The methods of delivery vary, but have included email transfer of data, FTP downloads and, more recently, retrieval through the web. Each primary database consists of a number of sequence files, which are text files formatted in a particular sequence format. A sequence file usually contains data for a number of protein sequence entries, with each such sequence entry providing a description of the protein and its amino acid sequence (see Figure 2 for an example).

For example, the Protein Data Bank (PDB) [10], which was established in 1971, is an international repository for the processing and distribution of experimentally-determined structure data that is growing dramatically. Similarly, SWISSPROT is a curated protein sequence database with a high level of annotation of protein function, and is supplemented by another database, TrEMBL [6]. Many other such sequence databases are also available (eg. the PIR International Protein Sequence Database [8]), and a list of more than 200 of the main biological databases was recently compiled [9]. The sheer number and variety indicates the importance of wrapping them up and providing them to the agent community in a uniform manner, but we will not provide an exhaustive list here.

The primary database agents manage local copies of different primary databases, with one agent per database. Currently, all the primary databases are obtained from FTP locations: a primary database

agent queries the appropriate FTP site on a regular basis to see if any of the files making up the database have changed. If so, the agent determines the changes that are required to the entries in its copy of the database to update it to a current version. Other agents can then query the primary database agent to obtain information about a particular primary database, the files making up the database or the individual sequence entries in the database. Agents can also subscribe to this information if they want to be notified when it changes.

### 3.3 Non-Redundant Database Agents

A non-redundant database (NRDB) is one constructed from a number of primary databases with entries that are deemed to provide duplicate information excluded. There are three main categories of non-redundant database, storing DNA sequences, protein sequences and protein structures respectively. Further classifications of non-redundant databases may result from considering the number and quality of primary databases used in the construction of the non-redundant database, or from the specifc notion of *redundant information* that is used. For example, some non-redundant protein sequence databases only regard sequences as redundant if they have identical residue strings, while others regard sequences as redundant if they have more than 90% of residues identical. Each type of non-redundant database has its specific uses in bioinformatics.

Now, a single agent in the GeneWeaver community manages each different type of non-redundant database. In the initial prototype community, there is only one NRDB agent, which constructs a non-redundant database from protein sequences held by all the primary database agents in the community using *identical* residue strings as the definition of redundancy. Hence the non-redundant database contains no sequences that have residue strings identical to any other sequence in the non-redundant database. Each of these non-redundant sequences is annotated with database cross-references indicating which sequence in the primary database has the most detailed annotation for this residue string.

In general, an NRDB agent subscribes to agents managing primary databases from which it wishes to derive its database. It builds up the non-redundant database by adding any new entries that have been added or modified in the primary databases since the non-redundant database was last updated.

### 3.4 Calculation Agents

Calculation agents apply particular skills to try to identify the structure or function of particular sequences of genomic data. A variety of techniques are used in a range of available programs, including similarity (homology) searches of greater or lesser sensitivity (eg. BLAST [2, 3] and FASTA [34, 35]), the use of sequence motif patterns (eg. InterPro [5]), sequence alignment (eg. ClustalW [37]), secondary structure prediction (eg. PhD [36], PSIPRED [23]), membrane topology prediction (eg. MEMSAT [25]), fold recognition (eg. Threader [24], GenThreader [22]), and so on. For example, BLAST (Basic Local Alignment Search Tool) is a set of rapid similarity search programs that explore all available sequence databases [2]. The search can be performed by a remote server through web pages or have results returned by email, or on a local machine.

Now, some of the methods take longer to run than others, while some provide more accurate or *confident* results than others. In consequence, it is often necessary to use more than one of these tools depending on the results at any particular stage. While a high confidence is desirable, time-consuming methods should be avoided if their results are not needed.

Although these tools already exist, they are largely independent. Calculation agents provide a way to integrate their operation in support of appropriate combinations of methods to generate confident results, and integrate them with, and apply them to, the accumulating genome databases. In short, the

tools are encapsulated in an agent wrapper so that existing applications such as BLAST can become independent agents in the GeneWeaver community.

A second consideration when managing these methods is the presence of dependencies between the methods. For instance, different sequence similarity web servers may be asked to search the SWISSPROT database, but each server will probably be searching a different version of the database, returning results that may refer to different entries. A more complicated dependence results when one program actually uses another internally. For example, the MEMSAT method uses an internal version of the BLAST method during its processing. The version of BLAST used, and the version of its underlying database, is hidden within the MEMSAT method itself. These underlying dependencies between the different methods can result in subtle inconsistencies in the different results.

In order to deal with this consistency problem, the calculation agents attempt to make these dependencies explicit. Whenever possible, a calculation agent constructs the underlying databases or parameters that its encapsulated analysis techniques depend on, using the data and methods available from other agents in the community. For instance, the BLAST calculation agent shown in Figure 3 uses the NRDB agent's non-redundant database to construct the database it searches. Similarly, the MEMSAT agent constructs training parameters for its method by using membrane topology sequences held by the SWISS agent. The MEMSAT agent also uses the BLAST agent when it needs to carry out the BLAST method during its internal processing. These interactions, shown in Figure 3, result in a system that has an additional degree of data and method integration beyond just externally combining the results obtained from different methods. (In addition, since the MEMSAT agent re-trains itself when new membrane topology sequences are added to the SWISSPROT database, its ability to predict membrane topologies for proteins improves over time. This can be seen as a rather specialized form of learning using data and methods that become available in the community.)

### 3.5 Genome Agents

Each genome agent manages the genome information for a particular organism. This can consist of the DNA sequences making up the genome of the organism together with the protein sequences expressed by the DNA sequences. Data such as known protein homologues and other derived results are stored together with their relationship to the nucleic acid or protein sequences. The primary data for the genomes, either DNA or expressed protein sequences, is obtained through FTP from a primary data source. Like the primary databases, these are under the control of third-parties and may be modified at any time. Genome agents check these sources of data on a regular basis, update their data accordingly when any changes occur, and apply the methods available in the community to annotate the genomes with functional and structural information. These derived results also need to be managed, and possibly recalculated, when the underlying genome data is modified by third-parties, or when the version of the method employed is updated.

## 4 Agent Architecture

Each agent in the GeneWeaver community shares a common architecture that is inspired by, and draws on, a number of existing agent architectures, such as [21], but in a far more limited and simplified way. An agent contains a number of internal modules together with an external persistent data store that is used for the storage of data it manipulates and external programs used for sequence analysis. In this section, we briefly describe the generic modules that comprise the architecture illustrated in Figure 4, which forms the basis of each agent. This will be used to provide a context for the description of inter-

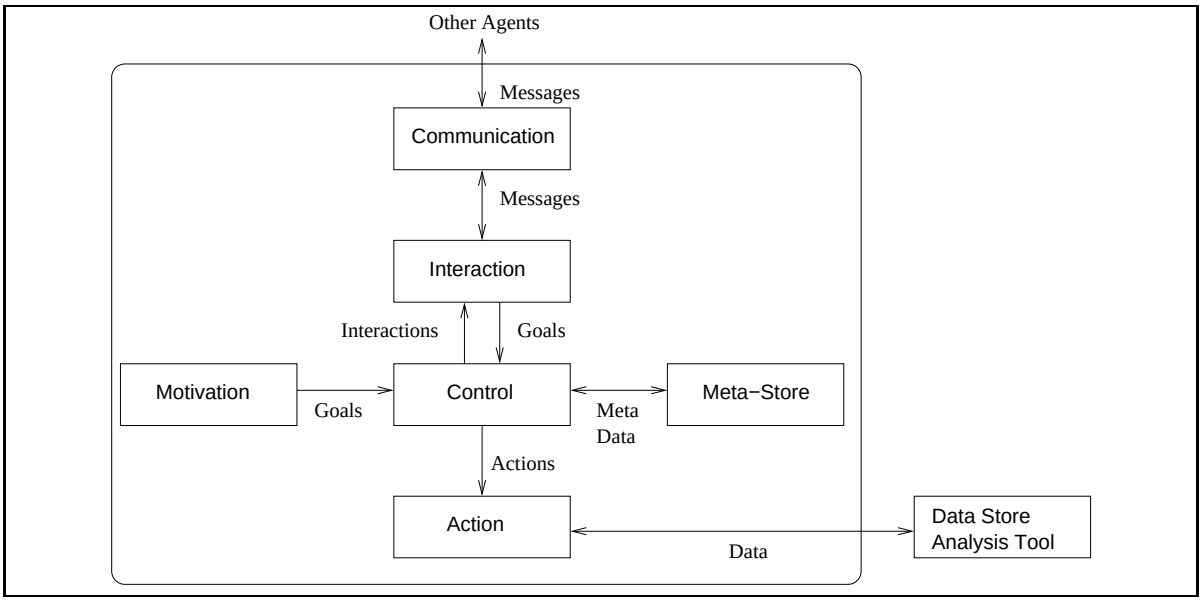


Figure 4: GeneWeaver agent architecture.

agent communication given in the following section. A more extensive description of the internal structure of the agents is presented in [11].

## 4.1 Motivation Module

Since each agent in the GeneWeaver community has different responsibilities and is required to perform different tasks, distinct high-level direction must be provided in each case to cause essentially the same architecture to function in different ways. The conceptual organisation of GeneWeaver agents thus involves the use of some high-level *motivations* that cause *goals* specific to the agent's tasks and responsibilities to be generated and performed [30, 31]. An agent is therefore initialised with the motivations required to carry out its responsibilities effectively. For example, all agents (except brokers) are motivated to register themselves with a broker agent, and will generate goals and actions to do so until they have succeeded. Similarly, a primary database agent has a motivation to cause the generation of specific goals and actions to update the primary database on a regular basis in order to ensure that it is up-to-date; and a non-redundant protein agent is motivated to subscribe to a broker for information on primary database agents and to subscribe to these latter agents for all relevant information.

## 4.2 Control Module

Perhaps the most important of the components of the agent architecture is the *control module*, which organises how the agent should pursue the satisfaction of *goals*. It decides whether a goal should be accomplished using a local *skill* of the *action module*, whether the goal should be broken down into a number of sub-goals using a *plan*, or whether it should be carried out by the *interaction module* since it involves either the provision of assistance to another agent, or the request of assistance. In order to do this, the information contained in the *meta-store* is used to decide how best to carry out actions, for example by an agent examining its own skills and that of other agents. In addition, the *control module*

determines whether the information in the *meta-store* should be modified as a result of performing an action.

### 4.3 Action Module

The *action module* is responsible for managing and performing *skills* that modify the underlying data being manipulated by the agent. The resulting actions typically involve performing operations on some input data and may (optionally) result in some output. In this respect, they are likely to use and modify data in the agent's data store. The action module is thus critically important in terms of the agent functionality in this domain, since it is the only module that can interface to the data store and thus provides the only way to modify the data the agent is working with. (In the case of calculation agents, the same would apply to the analysis tools instead of the databases.) Each agent's action module is instantiated with a number of *skills* (or types of action) that the agent can perform. The *action module* is invoked by the *control module*, and can also cause the *control module* to invoke message-passing actions in the *interaction module* when needed as part of an overall task. This might occur when another agent is needed to perform an action for the overall plan.

### 4.4 Interaction and Communications Modules

The *interaction module* handles the higher level interaction between different agents. Several possible types of *interaction* exist, each of them following a particular fixed interaction protocol. The interactions are key to agent cooperation and will be extensively described in the next section.

The interaction module is initialised with the types of interaction it can service (or the types of provider actions it can initiate). For example, the broker is the only agent that initiates its interaction module with a *register* action to service requests to *register* with it from other agents. The particular actions an agent is able to provide to others are recorded in its meta store so that it has an accurate picture of its capabilities.

Now, the communications module for one agent interacts with communications modules of other agents, using particular transport protocols. It also passes messages on to the agent itself for interpretation and processing, as well as accepting outgoing messages to be sent out to others. In this way, one agent interacts with another through their respective communications modules.

The mechanics of the interaction of agents in the community is achieved through message-passing communication using a number of transport protocols such as RMI [15] and CORBA [38], which is handled by the *communications module*.

### 4.5 Meta Store

The *meta-store* simply provides a repository for the information that is required by an individual agent for correct and efficient functioning. It contains relevant information about the types of *skills*, *plans* and *interactions* supported by different agents within the community. For example, the meta-data contained in this repository will enumerate the properties and capabilities of the agent, including aspects such as the protocols the agent can use, the skills that can be executed, and the agent's motivations. As other modules are instantiated on initialisation, this information is added to the *meta-store*. Thus, as the *action module* is initialised, the skills that can be performed by it are added to the repository.

The *meta-store* also provides a representation of the other agents in the community in order to determine how best to accomplish particular tasks, possibly using other agents. Information contained in it may be extended while the agent is running so that additional or newly-discovered information

| BAMO type | Agent Property |
|---|---|
| AgentInfo | General agent features |
| TransportInfo | Transport communication protocols supported |
| LanguageInfo | Content languages supported |
| OntologyInfo | Content ontologies supported |
| RequesterInfo | Interaction types the agent can request |
| ProviderInfo | Interaction types the agent can provide |
| SkillInfo | Skills that may be employed |
| PlanInfo | Plans that may be employed |
| MotivationInfo | Agent motivations |
| RelationInfo | Current ongoing relations with others |

Table 1: Types of meta-data present in the BioAgent Meta Ontology

about itself or other agents may be included. The only significant interaction is with the *control module* which records information in the *meta-store* as appropriate, and also uses it in decision-making.

# 5 Agent Communication

## 5.1 The BioAgent Language (BAL)

Agents in the GeneWeaver community communicate with each other using the BioAgent Language (BAL), which is constructed along the lines of KQML [32], with similar performatives, but is both dramatically reduced in range, and tailored to the particular kinds of interactions likely in GeneWeaver. Specifically, the data content of the BAL messages is vital, since the data is the key resource in the system, and it is extremely large in quantity. In this section, we describe BAL and show how it is used by agents for critical interactions.

BAL *interactions* lie at the highest level of communication, and consist of a number of messages being passed between agents following a fixed protocol for the particular type of interaction. Within an *interaction*, specific communication between agents is accomplished by sending BAL messages, which follow a simple speech-act structure by which a performative is associated with a particular kind of *content*. Depending on the performative, the *content* may consist of an ordered list containing query expressions, *data*, *meta-data* or simple strings. BAL messages employ *language* and *ontology* fields to indicate the language used for the content and the meanings assigned to symbols used.

The BioAgent Content Language (BACL) is a very simple language used to represent meta-data, data and query expressions. The BioAgent Meta Ontology (BAMO) defines the different types of meta-data (Table 1) and their meanings. This is the information used by the agent community to manage and organise itself, more details of which can be found in [11]. The BioAgent Data Ontology (BADO) defines the data types employed (Table 2). These are simply the major artefacts in this domain.

BAL messages consist of a performative together with some data fields that can include the sender and receiver, transport protocol, content language, content ontology, interaction reference string, page count and *content*, as shown below. The performative, sender id, receiver id, transport protocol, and reference string are all compulsory fields within a message, while the presence of the language, ontology, page count and *content* depends on the performative of the message. The list of performatives used by the GeneWeaver system are enumerated in Table 3.

| BADO type | Description |
|---|---|
| RemoteDB | Remote primary database |
| LocalDB | Local database |
| DBFile | Sequence database file |
| DBSeqEntry | Sequence database entry |
| Protein | Protein sequence |
| MemTopol | Protein membrane topology |
| Homologue | Sequence homology data |

Table 2: Types of data present in the BioAgent Data Ontology.

```
Sender:    <sender id>
Receiver:  <receiver id>
Transport: <transport protocol>
Language:  <content language>
Ontology:  <content ontology>
Perform:   <performative>
Ref:       <reference string>
Page:      <page number>
Content:   <content>
```

The *sender id* and *receiver id* are in the form `Swiss007f000001`, which specifies the name of the agent (in this case SWISS) followed by 10 hexadecimal digits. The last 8 digits specify the IP address of the physical machine the agent resides on, with the first two digits specifying a particular *office* (0 to 100) on the machine. Each agent must have a unique name within a particular office. Essentially, an office is a running process that permits a number of agents to be placed within it, each agent existing as a number of threads within the process. In our current implementation, each office is a Java Virtual Machine, and it permits a variety of configurations. A large number of agents may be placed into a single office for a small memory footprint, although this makes them more dependent on each other if, for example, one of them causes a thread deadlock. By using a separate office for each agent, the community is more robust to failure in any one agent, but this uses correspondingly more resources.

Currently three *transport protocols* are used: `rmi`, `corba` and `direct`. RMI is the default protocol supported by all the agents within the Java implementation of GeneWeaver, while CORBA allows agents implemented in different languages to be supported as long as they conform to the BAL language specification. Such agents, running under a different management system, would need to be allocated their own range of office numbers. For example, 101 to 150 could be allocated for agents running under a C++ management system. The `direct` protocol is used to communicate between agents in the same office (or process) simply by using synchronized methods. It permits the configuration in which a community of agents can cooperate in the same office (including a broker) without the need for any network communication, and allows agents to make their services available only to other agents in the same office, providing a secure form of agent hiding.

A unique *reference string* is generated by the agent initiating an interaction as a means of identifying and keeping track of the course of that interaction, especially when there may be several concurrent interactions taking place. It is used in all messages that are part of the particular interaction. The optional components of BAL messages are determined by the kind of message (and hence performative) under consideration.

The *language* and *ontology* fields permit agents to handle multiple content languages as appropriate and different data *contexts* within these languages. The bioinformatics community employs a

| Performative | Description |
|---|---|
| `ask` | Sender wishes to know if any data exists in the receiver's data store (or meta-store) which matches the given data query. |
| `tell` | Indicates that the given data exist in the data store (or meta store) of the sender. |
| `deny` | Indicates that the data given no longer exists in the data store (or meta store) of the sender. |
| `subscribe` | Sender wishes to subscribe to data (or meta-data) matching the given query and be informed about future changes. |
| `unsubscribe` | Remove the subscription. |
| `do` | Sender wishes the receiver to carry out the specified skill with the given content as input. |
| `derive` | Sender wishes the receiver to derive data satisfying the given data query using its plans and skills. |
| `ok` | Confirms an action has been successful. |
| `error` | Terminates a conversion since either an agent did not understand the contents, or the message protocol was not followed. |
| `sorry` | Terminates a conversion. The agent understood the message but did not have any response to it. |
| `register` | Registers an agent with the broker. This provides the broker with a symbolic name and a description for the agent. |
| `unregister` | Cancels a register (and any commitments of the agent). |

Table 3: Description of BAL performatives.

wide-range of languages to represent data and data queries, including SQL, XML, XQL and ASN-1 [1], and several ontologies are being developed for use within different sub-fields [7, 13]. Currently, however, we employ a simple content language and simple ontologies, although it is clearly beneficial to permit different languages and ontologies to be used within specialized sub-communities of agents.

The *page count* is used, for example, when a long list of data items which cannot efficiently (or appropriately) be included in a single message, needs to be sent between agents, and it may be necessary to split it into multiple messages. With vast amounts of data being generated daily in this domain, this is not an unlikely event. The need to use a page count to split data among messages in this way applies only to *tell* and *deny* performatives when used to transfer large amounts of data.

Now, the contents of the messages vary according to the kind of performative used, and are enumerated in Table 4. These may be ordered lists of query expressions, *data*, *meta-data* or simple strings.

## 5.2 BAL Interactions

A BAL interaction represents a communicative episode at the highest level of abstraction. It consists of a number of messages sent between two agents following a fixed protocol for the type of interaction. All the messages are required to share the same reference string to indicate that they are part of the same interaction. In the discussion below, all interactions concern only two agents. The agent that requests assistance (and hence the interaction) is denoted the *requester* and the agent providing assistance is denoted the *provider*.

| Performative | Content |
|---|---|
| `ask` | Data or meta-data query. |
| `tell` | List of data or meta-data items, page number required. |
| `deny` | List of data or meta-data items, page number required. |
| `subscribe` | Data or meta-data query providing what to subscribe to. |
| `unsubscribe` | No content (reference string used to determine what to unsubscribe). |
| `do` | Meta-data specifying the *Skill* to do, followed by a list of data specifying the input. |
| `derive` | Data query specifying the required data to derive followed by a list of data which may be used. |
| `ok` | No content. |
| `error` | String giving reason for error. |
| `sorry` | String giving reason for failure. |
| `register` | AgentInfo meta-data for the agent registering. |
| `unregister` | No content (reference string used to determine what to unregister). |

Table 4: Content types of different performatives.

An example of registration is as follows. The SWISS agent (which is a primary database agent) registers itself with the BROKER by sending a *register* message as follows.

```
Sender:    Swiss007f000001
Receiver:  Broker007f000001
Transport: rmi
Language:  BACL
Ontology:  BAMO
Perform:   register
Ref:       Swiss007f000001_0
Content:
AgentInfo(
  ID = 0,
  OWNER = Swiss007f000001,
  TYPE = PrimaryDB,
  MOD_TIME = 2001-04-04T19:40:35+0100
  LOCAL_MOD_TIME = 2001-04-04T19:40:35+0100
  AGENT_URL = "http://insulin.brunel.ac.uk/~bryson/agents/swiss/index.html",
  DESCRIPTION = "Swiss agent which manages the SWISSPROT database."
)
```

The content of the message consists of a single meta-data item of type AgentInfo that includes ID, OWNER, TYPE, MOD_TIME and LOCAL_MOD_TIME fields. The OWNER field identifies the agent that *owns* the meta-data item, while ID provides a unique identifier for the item within that particular agent. The MOD_TIME field gives the time, in ISO format, of the last change to the actual data item (as held by the owner), whereas LOCAL_MOD_TIME gives the time the item was last changed in the local agent (since agents may have copies of data owned by other agents). In addition, there are also AGENT_URL, DESCRIPTION and TYPE fields to provide further information about the particular agent, such as the location of its home web page. If the BROKER permits registration, it will reply with the following *ok* message.

```
Sender:    Broker007f000001
Receiver:  Swiss007f000001
Transport: rmi
Perform:   ok
Ref:       Swiss007f000001_0
```

## 5.3 Interaction Scenarios

We have described the modular framework of an agent together with the language used for inter-agent communication. In this section, we show how these aspects come together to enable effective agent operations for the management and integration of data and methods within the agent community. We describe four general scenarios that result from just setting up the appropriate motivations, interactions, skills and plans in the agents. In each scenario, the goals adopted by agents on either side of the interaction are described together with the protocol followed and the messages sent as part of that protocol. The complete interactions are illustrated with state transition diagrams, which come in parts: one for the requester and one for the provider. The diagrams are largely self-explanatory, and serve to illustrate the discussion, so only a cursory explanation of them (as opposed to the interactions) is given.

### 5.3.1 Registering with a Broker

Each agent (other than brokers) has a motivation to register with a broker, which will lead to the generation of a *goal* with the aim of forming a register relationship with a broker agent. (We denote goals with the aim of making or breaking relationships between agents as *relationship goals.*) The control module satisfies this goal by examining the meta-store for information on a suitable broker, and then initiating a *register* interaction using the interaction module. This causes a *register* message to be sent to the broker agent via the communications module, and the agent then waits for a response. For example, the primary database agent, SWISS agent, which manages the SWISSPROT database, sends the *register* message above to the broker agent.

On receiving the *register* message, the broker starts its side of the interaction, resulting in a *register* relationship goal being passed to its control module. In turn, the control module satisfies this relationship goal using an appropriate *plan*, which causes the broker agent to *subscribe* (see next section) to relevant *meta-data* of the registering agent such as the agent's *skills*, *databases* and *plans.* This allows the broker to keep track of any relevant changes in information about the agent, such as when one of its skills is updated. In addition, the broker agent updates its *meta-store* to include details of the registered agent. If the registration *plan* is successful, this will result in an *ok* message being returned by the broker's side of the interaction (as shown above).

Finally, on receiving an *ok* message from the broker, the registering agent's *relationship goal* is satisfied, with the intensity of the relevant motivation to register being lowered since it is now satisfied. This reduction in intensity enables other motivations of the agent to become active.

Alternatively, the broker may not allow the agent to register. For instance, a security policy may only allow agents running on certain hosts to register, or the community organized by the broker may be of a specialized type that does not include primary database agents, In such cases, a *sorry* reponse might result. This interaction scenario is illustrated by Figure 5.

```
Sender:    Broker007f000001
Receiver:  Swiss007f000001
Transport: rmi
Perform:   sorry
Ref:       Swiss007f000001_0
Content:
Invalid agent type
```

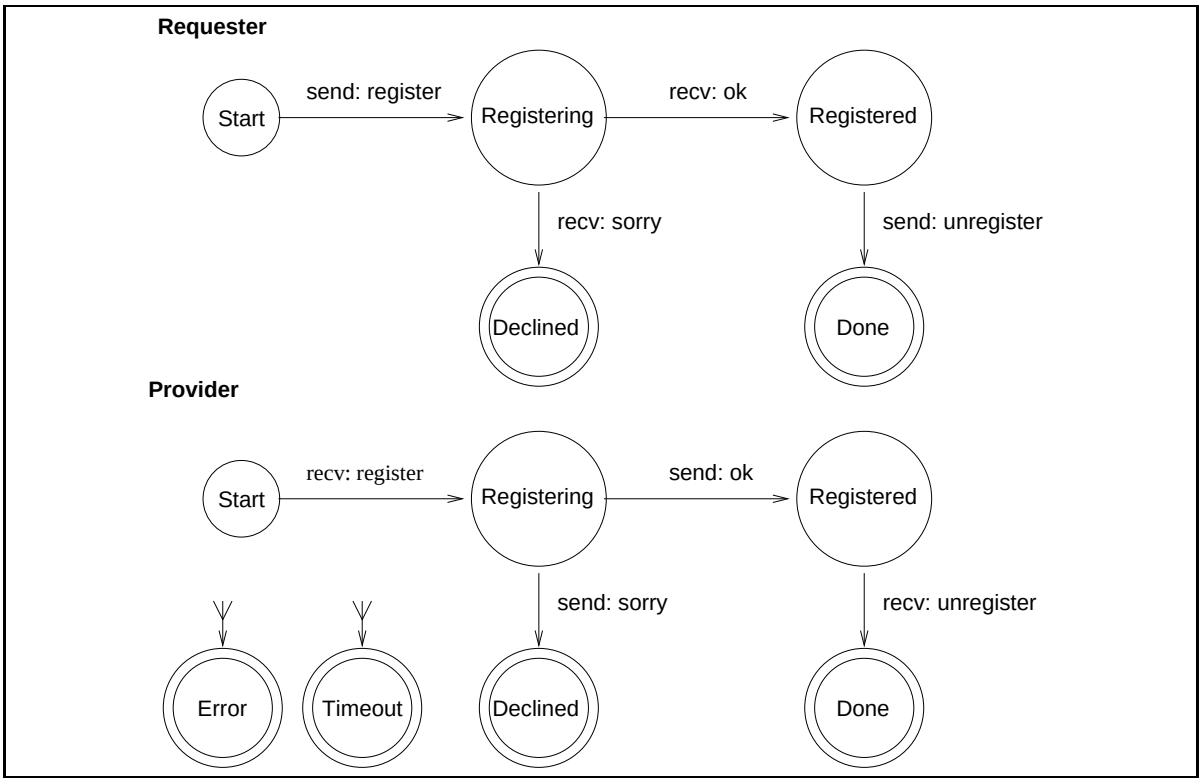


Figure 5: State transition diagram for the registration interaction protocol.

### 5.3.2 Broker Subscription Scenario

The non-redundant database (NRDB) agent is motivated to subscribe to the broker for information about all agents owning local primary databases containing protein sequences. This motivation is typically activated (through having the highest intensity level) once the agent has successfully registered, and causes a *relationship goal* of type `subscribe` to be sent to the control module. The control module satisfies this *relationship goal* by querying the meta-store for all brokers with which it is currently registered. For each broker, the control module creates a *subscribe* interaction which, in turn, results in a *subscribe* message being sent.

```
Sender:    NRDB007f000001
Receiver:  Broker007f000001
Transport: rmi
Language:  BACL
Ontology:  BADO
Perform:   subscribe
Ref:       NRDB007f000001_21
Content:
LocalDB(
  ID?, OWNER?, SOURCE?, QUALITY?,
  TYPE?, NAME?, MOD_TIME?,
  LOCAL_MOD_TIME?, DESCRIPTION?
)
```

On receiving this *subscribe* message, the broker responds by starting its side of the interaction. If the broker refuses to form the relationship, then it responds with a *sorry* message, otherwise it responds with an *ok* message to indicate that the relationship has been accepted. This causes a query

to be performed on its meta-store on a regular basis to provide the required information, resulting in *tell* and *deny* messages being sent back to the requester, giving information about currently registered agents, and those that have deregistered. Such periodic queries continue until the interaction is halted on receipt of an *unsubscribe* message from the requester. The following example message illustrates the kind of response that might be received in this way from the broker.

```
Sender:    Broker007f000001
Receiver:  NRDB007f000001
Transport: rmi
Language:  BACL
Ontology:  BADO
Perform:   tell
Ref:       NRDB007f000001_21
Page:      0
Content:
LocalDB(
  ID = 11,
  OWNER = Swiss007f000001,
  SOURCE = SWISSPROT,
  QUALITY = 90,
  TYPE = PROTEIN,
  NAME = SWISSPROT,
  MOD_TIME = 2001-04-04T20:07:21+0100,
  LOCAL_MOD_TIME = 2001-04-04T20:08:11+0100,
  DESCRIPTION = "SWISSPROT manually annoted protein database"
)
```

Later in the session, a *deny* message may be sent if, for example, the SWISS agent deregisters from the broker.

Back at the requester's end, *tell* goals are instantiated as a result of any received *tell* and *deny* messages, and cause the *control module* to carry out an appropriate *plan*. In this case, the result is to add information to the meta-store, or delete information from the meta-store as appropriate. This interaction scenario is illustrated by the state-transition diagrams of Figure 6.

### 5.3.3 Subscribe Interaction Scenarios

The *subscribe* interaction is also used in a number of other places in the prototype, such as when the non-redundant database (NRDB) agent is motivated to subscribe to primary database agents for sequence data. Typically, this motivation becomes active after the agent has successfully subscribed to the broker for information on agents owning local primary databases, so that the intensity of that motivation decreases. The NRDB agent then forms *subscribe* interaction relationships with each of these agents to obtain information about their primary databases. The subscription remains in force while both agents are present in the community, allowing the NRDB agent to maintain its database with the latest information known throughout the community.

Calculation agents also use the *subscribe* interaction. The BLAST agent subscribes to the NRDB agent so that it can ensure that the database it searches with BLAST is up-to-date. Similarly, the MEMSAT agent subscribes to the SWISS agent for information about membrane proteins so that it can keep its method trained with the most recent data.

In general, subscription to another agent for particular types of data may be regarded as a *data push* protocol, in that new or modified data is pushed onto the requester agent as it becomes available. The *tell* interaction, which is not used within the prototype, is an unsolicited form of *data push*. When an agent has data it believes another agent may be interested in, it simply sends a *tell* message to this agent informing it about the data, whereupon it receives an *ok* or *sorry* response informing it whether the other agent is interested in the data provided.

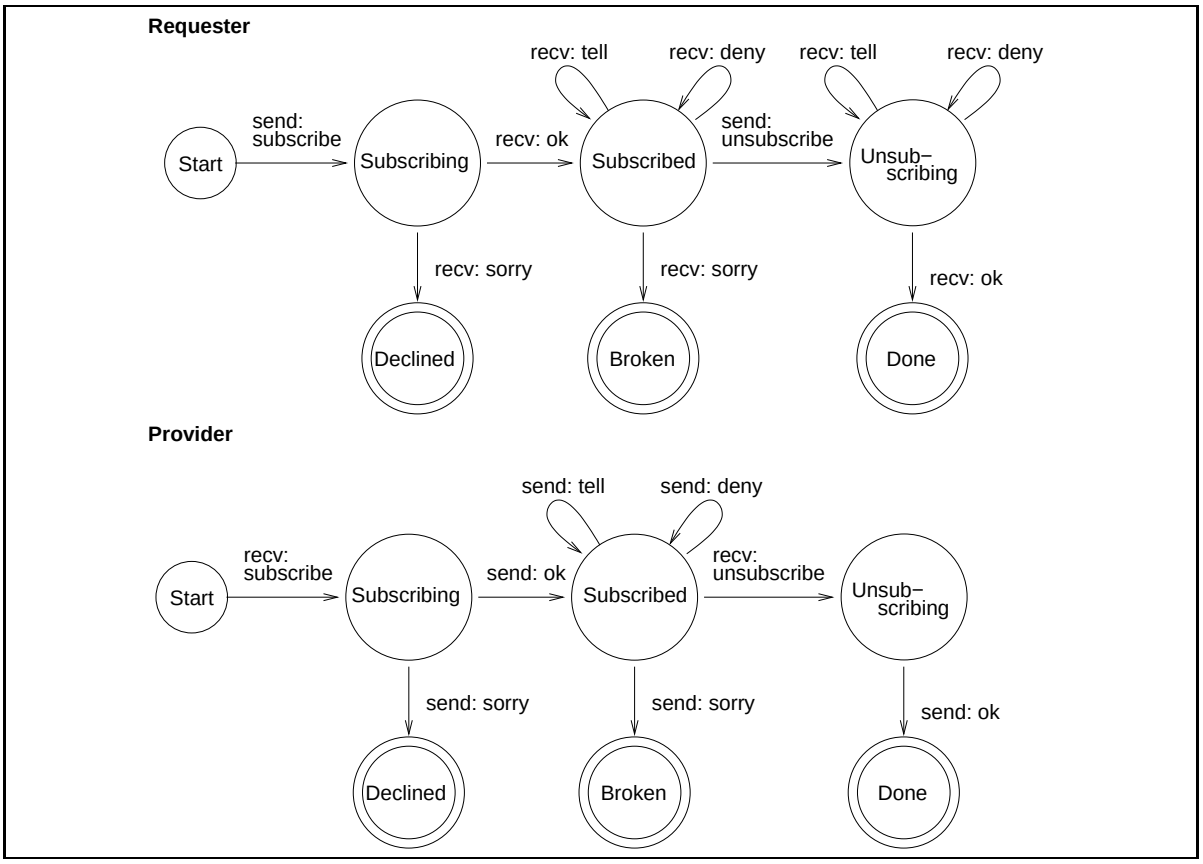


Figure 6: State transition diagram for the subscription interaction protocol. The *Error* and *Timeout* states are not shown for clarity.

### 5.3.4 Calculation Agent Requests

The main concern of previous scenarios has been the manipulation of data and meta-data. However, agents also have *plans*, involving *sequences* of action, that may be used to satisfy goals by either manipulating the agent's meta-store or by decomposing a goal into a number of sub-goals which may then be tackled. This section deals with scenarios involving interactions that use *skills* combined with *plans* to satisfy goals.

Suppose a genome agent is initiated with a number of motivations: registering with a suitable broker, subscribing to the broker for information about the *skills* and *plans* available in the community, updating its genome data using an external data source and annotating this genome data. Now, deriving different types of annotation is brought about by different motivations. For example, the motivation to determine homologues for the proteins expressed by the genome provides functional annotation, while the motivation to determine membrane topologies for these proteins provides a form of structural information.

We will examine the motivation to determine membrane topologies in more detail since this involves both *derive* and *do interactions.*

Generally, a genome agent will not know the best approach to determine a membrane topology for a protein in the genome it is annotating. Thus the motivation generates a *derive* goal with the aim of deriving a membrane topology with a suitable quality for the particular protein. Now, the *control*

*module* determines how to satisfy this goal by inspecting the meta-store. Suppose that it finds that it does not have a local *skill* or *plan* itself to satisfy deriving a membrane topology, but is aware of another agent in the community, the MEMSAT agent, which contains a suitable *plan.* To utilise this plan, the control module creates a *derive* interaction, resulting in a *derive* message being sent to the MEMSAT agent.

```
Sender:    HInf007f000001
Receiver:  Memsat007f000001
Transport: rmi
Language:  BACL
Ontology:  BADO
Perform:   derive
Ref:       HInf007f000001_42
Content:
MemTopol(
  N_LOC?, OTHER_LOC?, N_LOC_SIDE?,
  SPANS?, QUALITY?, DERIVED?,
  SEQ_REF = 74_HInf007f000001
)
```

On receiving this message, the interaction module of the MEMSAT agent creates a *derive* goal that is sent to its control module which, in turn, finds a suitable *plan* to accomplish the goal and executes it. Now, this *plan* is fairly complicated and involves three stages. The first involves obtaining the residue string of the sequence to analyse if this is not already provided. This can be satisfied by a *query* interaction (described later) to obtain the information from the genome agent. The second stage involves creating a *do* goal requesting that the BLAST skill be used to determine a sequence profile for the sequence. Since this is a well-specified task, it is satisfied by using a *do* interaction to request the BLAST agent perform the calculation. Finally, a second *do* goal requesting that the MEMSAT method be applied to this profile generates the result, which is satisfied by using the MEMSAT *skill* local to the agent. If all the stages of the plan are successful, the original *derive* goal will be satisfied, causing the MEMSAT agent to send a *tell* message providing the result. The genome agent then uses this information to annotate the sequence.

Both the *derive* and *do* interaction protocols are presented in Figure 7. This diagram also shows the *query* interaction, which involves an initial *ask* message, but is otherwise very similar to the others. It simply allows a one-off query of the data held by another agent, without receiving further updates; it is useful in the scenario above when the initial *derive* goal does not contain all the required information for the *plan.*

Note that these protocols may generate a number of pages of results, with a page number of zero indicating the last page.

## 5.4 Summary

In describing these sample interactions, we show how the system manages and integrates the distributed databases, skills and expertise (encoded as plans) of agents within the community. It may be seen that a number of categories of interaction are involved. The *subscribe* and *tell interactions* provide *data push* protocols while the *query interaction* provides a *data pull* protocol. The *do* and *derive* interactions are imperative and request action, the *do* interaction being used when the requester agent knows exactly which *skill* it wishes to apply, and the *derive* interaction being used otherwise, leaving it to the provider to decide. Finally, the *register* interaction is used to populate and manage the agent community. Though we have only described some examples of interaction within the prototype, the overall nature of the system interactions should be clear, with brokers providing much of the capability for organising large numbers of databases and services. Certainly, there are many other issues here

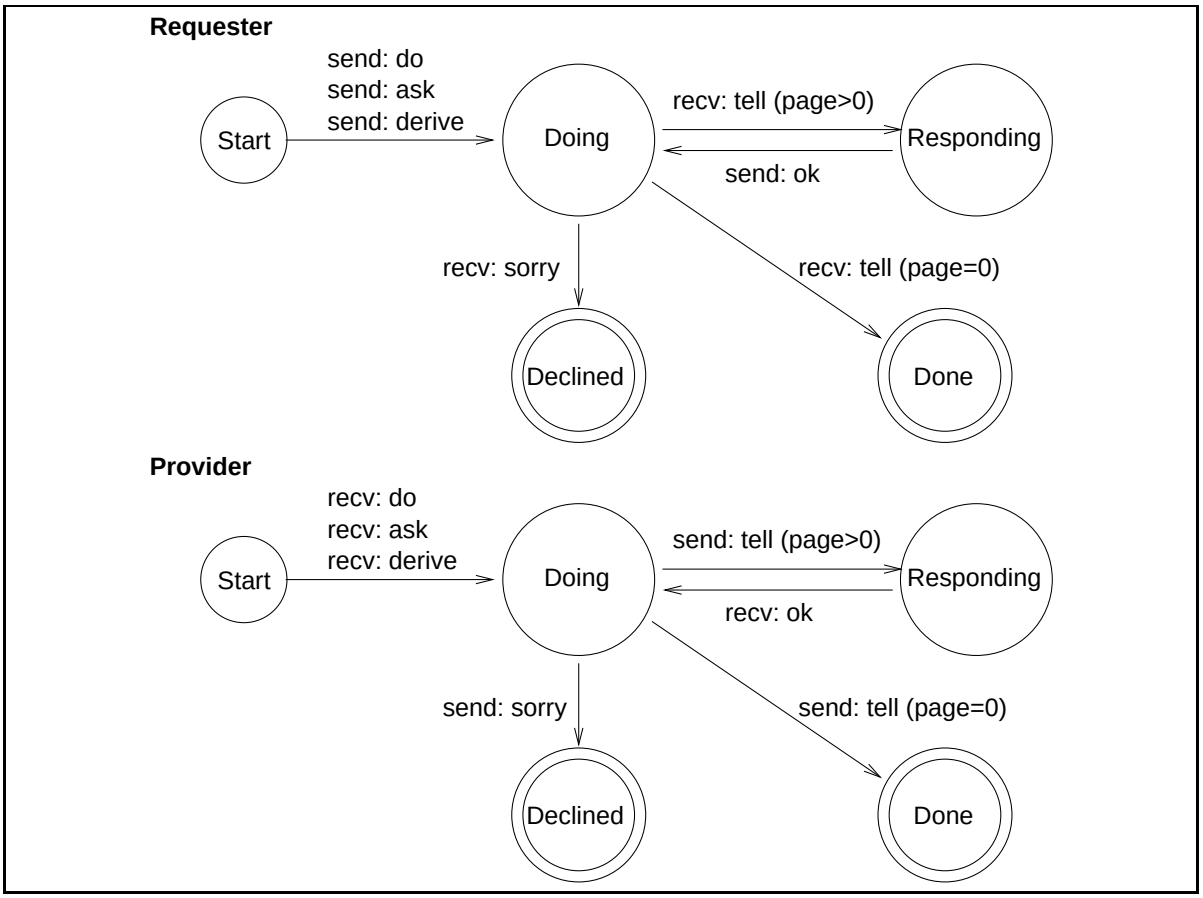


Figure 7: State transition diagram for the do, derive and query interaction protocols. The *Error* and *Timeout* states are not shown for clarity.

that need to be considered (for example in relation to the potential use of different data languages [1] and ontologies [7, 13] to provide a consistent language for the description of biological data) but these are separate concerns and we do not consider them further here. It should be clear though that since the BAL interaction language is at a higher level it permits different underlying content languages to be employed.

# 6 Discussion and Conclusions

The problems raised by the surge of information being added to biological databases demand a significant effort in order to solve them. This is particularly important because of the potential benefits that an understanding of the information locked away in these databases might bring. In this paper we have introduced the GeneWeaver agent community for just this purpose, and described the architecture of the individual agents within it. Concentrating on the fundamental aspects of agent interaction, we have directed attention towards specific scenarios concerned with the management and integration of distributed databases and analysis methods.

Related efforts have taken different approaches to the same problem. For example, GeneQuiz [4] and PEDANT [17] are two functionally rich systems for genome analysis, which integrate a large

number of databases and analysis software using Perl code, in a largely monolithic fashion. The designs have been successful, but require that all the resources are held locally, raising possible concerns about extensibility and scalability.

Adopting a more distributed approach, MAGPIE [18] treats each genome as an individual project to which a number of remote or local methods may be applied. in a highly configurable manner. For instance, remote web servers may be used, distributing both the computational load and the management responsibilities for the underlying databases and tools. However, the data employed by most remote servers is consequently hidden so that there is potential for the data used by different servers to be inconsistent. External sites may have very different policies on how often they update their underlying data, resulting in variations in the quality of output from servers based on similar methods. Also, the updating of external servers is generally not notified to the system so that it may respond by recalculating appropriate data.

In essence, when using external servers, the management of the methods and data is hidden. This can be remedied by providing distributed systems in which the different servers become an integral part of the framework, as in EDITtoTrEMBL [33]. Its distributed *analysis modules* provide *meta-data* about their analysis methods, so that appropriate methods can be chosen, in a similar way to the *meta-data* used within GeneWeaver to choose appropriate agents.

Although EDITtoTrEMBL is similar to the approach we adopt, a number of key differences exist between the architectures. In particular, EDITtoTrEMBL has a single thread of control (the *dispatcher module*) which drives its *analysis modules* in essentially a client-server fashion. The *dispatcher* module has the single goal of extending the annotation of a TrEMBL formatted sequence using the *analysis modules*. While this suffices for the particular application, GeneWeaver provides a more generic and extensible framework in which future goals of the system may not be entirely anticipated. For this reason, each agent has its own thread of control, driven by its own motivations and goals, resulting in a system which has a more peer-to-peer architecture. This allows new agents joining the community not only to provide services to the community, but to bring new goals which may be achieved by the community.

Secondly, the systems differ in the nature of the interactions between the distributed components. In our terminology the analysis modules of EDITtoTrEMBL deal with one type of interaction, a *do interaction* with a DBSeqEntry as input, resulting in additional annotation being added to the DBSeqEntry. Essentially, the distributed components are dealing with one type of goal, to *apply a particular analysis method to a sequence entry*. By contrast, to facilitate a cooperative community the GeneWeaver architecture employs a much richer interaction language, with relationships that may persist over long periods of time, allowing agents within the community to benefit from interactions with others. For example, the MEMSAT agent can improve its analysis as the community of agents discover more examples of membrane proteins. In turn, the genome agents benefit from the MEMSAT agent being more accurate, since they use it for satisfying their annotation goals.

Agent systems that are heavily embedded in application to other scientific domains differ from much early work in the field of information agents, which achieved high visibility and was responsible to some extent for driving the agent field forward. For example, agents for traversing and searching the web, and email and news filtering agents exemplified a large body of work at one end of the spectrum. These kinds of systems were not the only agents being developed, but occupied a central position in the perception of the work being done. The application of techniques developed for these relatively well-defined problems, however, has transferred into more general areas, and in more sophisticated and extensive systems. Electronic commerce is just one example of the natural extension and elaboration of earlier work and its application to an exciting new domain of activity [19].

Perhaps more important to the long term prospects of agent systems, though, are application do-

mains that are less intuitively obvious possibilities for agent systems deployment, but no less deserving or appropriate. This is because the successful development and use of agent systems by those who are unaware of the hype, and less concerned with the issues of the technology *per se* but more concerned with the benefits that it delivers, is more likely to sustain the agent paradigm in the longer-term. In this respect, the work described in this paper on the application of agent technology to bioinformatics, in order to make sense of the vast amounts of data that are being generated at an ever-increasing pace and stored at globally distributed but accessible sites, demonstrates the suitability of the agent paradigm in yet another very different domain.

The particular value of these efforts to the agent community is in solving problems that have not been created by the very technology (or related technology) that is being used to solve them. These problems and domains are pre-existent and *decoupled* from the solutions, and consequently provide what might be considered an objective demonstration of the utility of agent systems. For the cynics — and there are many — this is an acid test. More importantly, perhaps, they address a fundamental need in the biological sciences.

### Acknowledgements

The work described in this paper is supported by funding from the Bioinformatics programme of the UK's EPSRC and BBSRC.

## References

[1] F. Achard, G. Vaysseix, and E. Barillot. XML, bioinformatics and data integration. *Bioinformatics*, 17:115–125, 2001.

[2] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215:403–10, 1990.

[3] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res*, 25:3389–402, 1997.

[4] M.A. Andrade, N.P. Brown, C. Leroy, S. Hoersch, de Daruvar A., C. Reich, A. Franchini, J. Tamames, A. Valencia, C. Ouzounis, and C. Sander. Automated genome sequence analysis and annotation. *Bioinformatics*, 15:391–412, 1999.

[5] R. Apweiler, T.K. Attwood, A. Bairoch, A. Bateman, E. Birney, M. Biswas, P. Bucher, L. Cerutti, F. Corpet, M.D. Croning, R. Durbin, L. Falquet, W. Fleischmann, J. Gouzy, H. Hermjakob, N. Hulo, I. Jonassen, D. Kahn, A. Kanapin, Y. Karavidopoulou, R. Lopez, B. Marx, N.J. Mulder, T.M. Oinn, M. Pagni, and F. Servant. The InterPro database, an integrated documentation resource for protein families, domains and functional sites. *Nucleic Acids Res*, 29:37–40, 2001.

[6] A. Bairoch and R. Apweiler. The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000. *Nucleic Acids Res*, 28:45–8, 2000.

[7] P.G. Baker, C.A. Goble, S. Bechhofer, N.W. Paton, R. Stevens, and A. Brass. An ontology for bioinformatics applications. *Bioinformatics*, 15:510–20, 1999.

[8] W.C. Barker, J.S. Garavelliand P.B. McGarvey, C.R. Marzec, B.C. Orcutt, G.Y. Srinivasarao, L.-S.L. Yeh, R.S. Ledley, H.-W. Mewes, F. Pfeiffer, A. Tsugita, and C. Wu. The PIR-international protein sequence database. *Nucleic Acids Research*, 27(1):39–43, 1999.

[9] A.D. Baxevanis. The molecular biology database collection: an updated compilation of biological database resources. *Nucleic Acids Res.*, 29:1–10, 2001.

[10] F.C. Bernstein, T.F. Koetzle, G.J. Williams, E.E. Meyer, M.D. Brice, J.R. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasumi. The Protein Data Bank: a computer-based archival file for macromolecular structures. *J Mol Biol*, 112:535–42, 1977.

[11] K. Bryson, M. Luck, M. Joy, and D.T. Jones. Applying agents to bioinformatics in GeneWeaver. In *Cooperative Information Agents IV*, volume 1860 of *LNAI*, pages 60–71, 2000.

[12] F.S. Collins, A. Patrinos, E. Jordan, A. Chakravarti, R. Gesteland, L. Walters, and the members of the DOE and NIH planning groups. New goals for the U.S. human genome project: 1998-2003. *Science*, 282:682–689, 1998.

[13] THE GENE ONTOLOGY CONSORTIUM. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[14] THE GENOME INTERNATIONAL SEQUENCING CONSORTIUM. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.

[15] J. Farley. *Java Distributed Computing*. O'Reilly, 1998.

[16] J.D. Foss. Brokering the info-underworld. In *Agent Technology: Foundations, Applications, and Markets*, pages 105–123. Springer, 1998.

[17] D. Frishman, K. Albermann, J. Hani, K. Heumann, A. Metanomski, A. Zollner, and H.W. Mewes. Functional and structural genomics using PEDANT. *Bioinformatics*, 17:44–57, 2001.

[18] T. Gaasterland, A. Sczyrba, E. Thomas, G. Aytekin-Kurban, P. Gordon, and C.W. Sensen. MAGPIE/EGRET annotation of the 2.9-mb drosophila melanogaster adh region. *Genome Res.*, 10:502–510, 2000.

[19] R.H. Guttman, A.G. Moukas, and P. Maes. Agent-mediated electronic commerce: A survey. *Knowledge Engineering Review*, 13:147–159, 1998.

[20] M.N. Huhns and M.P. Singh. Multiagent systems in information-rich environments. In *Cooperative Information Agents II*, volume 1435 of *LNAI*, pages 79–93, 1998.

[21] N. R. Jennings and T. Wittig. ARCHON: Theory and practice. In *Distributed Artificial Intelligence: Theory and Praxis*, pages 179–195. Kluwer Academic Press, 1992.

[22] D.T. Jones. GenTHREADER: an efficient and reliable protein fold recognition method for genomic sequences. *J Mol Biol*, 287:797–815, 1999.

[23] D.T. Jones. Protein secondary structure prediction based on position-specific scoring matrices. *J Mol Biol*, 292:195–202, 1999.

[24] D.T. Jones, W.R. Taylor, and J.M. Thornton. A new approach to protein fold recognition. *Nature*, 358:86–9, 1992.

[25] D.T. Jones, W.R. Taylor, and J.M. Thornton. A model recognition approach to the prediction of all-helical membrane protein structure and topology. *Biochemistry*, 33:3038–3049, 1994.

[26] P. Kitcher. *The Lives to Come: The Genetic Revolution and Human Possibilities.* Penguin, 1996.

[27] D. Kuokka and L. Harada. Matchmaking for information agents. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, volume 1188 of *LNCS*, pages 672–679, 1995.

[28] D. Kuokka and L. Harada. On using KQML for matchmaking. In *Proceedings of the First International Conference on Multi–Agent Systems*, pages 239–245. MIT Press, 1995.

[29] M. Luck. From definition to deployment: What next for agent-based systems? *Knowledge Engineering Review*, 14:119–124, 1999.

[30] M. Luck and M. d'Inverno. Engagement and cooperation in motivated agent modelling. In *Distributed Artificial Intelligence Architecture and Modelling: Proceedings of the First Australian Workshop on Distributed Artificial Intelligence*, volume 1087 of *LNAI*, pages 70–84, 1996.

[31] M. Luck and M. d'Inverno. Motivated behaviour for goal adoption. In *Multi-Agent Systems: Theories, Languages and Applications - Proceedings of the Fourth Australian Workshop on Distributed Artificial Intelligence*, volume 1544 of *LNAI*, pages 58–73, 1998.

[32] J. Mayfield, Y. Labrou, and T. Finin. Evaluation of KQML as an agent communication language. In *Intelligent Agents: Theories, Architectures, and Languages*, volume 1037 of *LNAI*, pages 347–360, 1996.

[33] S. Moller, U. Leser, W. Fleischmann, and R. Apweiler. EDITtoTrEMBL: a distributed approach to high-quality automated protein sequence annotation. *Bioinformatics*, 15:219–227, 1999.

[34] W.R. Pearson. Rapid and sensitive sequence comparison with FASTP and FASTA. *Methods Enzymol*, 183:63–98, 1990.

[35] W.R. Pearson. Effective protein sequence comparison. *Methods Enzymol*, 266:227–58, 1996.

[36] B. Rost and C. Sander. Improved prediction of protein secondary structure by use of sequence profiles and neural networks. *Proc. Natl. Acad. Sci. USA*, 90:7558–7562, 1993.

[37] J.D. Thompson, D.G. Higgins, and T.J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22:4673–80, 1994.

[38] G. Vossen. The CORBA specification for cooperation in heterogeneous information systems. In Peter Kandzia and Matthias Klusch, editors, *Proceedings of the First International Workshop on Cooperative Information Agents*, volume 1202 of *LNAI*, pages 101–115, 1997.