USING WEB AGENTS FOR DATA MINING OF FUNGAL GENOMES

Audrius Meškauskas
Alte Gfennstr. 22, CH-8600 Dubendorf, Switzerland (AudriusA@Bioinformatics.org).

We created an application called Sight, a Java™-based package that provides a user-friendly interface to generate and connect agents for automatic genomic data mining without requiring programming skills from the user. Sight was originally developed to automate analysis of the human genome and attempts to generate web agents for fungus-related Internet resources revealed that some of those resources use new methods of representing the information they report, and some servers returned multiple intermediate pages leading towards their response, which created difficulties for automated recovery of results. Consequently, it was not possible to use effectively the old version of Sight so this version of the application was adapted with a little additional programming, creating a new version for which these features of the fungal genome servers do not represent a problem. The new version of Sight (v. 3.0.0) that is tailored to servers carrying fungal databases is freely available for download from the project website at these URLs:

http://bioinformatics.org/jSight/ and http://jsight.sourceforge.net/index_SF.htm.

1. INTRODUCTION

As more genomes are sequenced and added to the publicly available databases the opportunities for data mining expand in terms of both the range of organisms available and the biological systems open to analysis. For the most part, these opportunities are still enjoyed by the relatively few people (or teams) who have the necessary combination of the biological expertise to frame the research questions and the programming skills to take full advantage of the database server resources that are available. No computer programming can yet supply the biological expertise, but many of the operations involved in data mining server databases are mechanical and repetitive. These are open to management by appropriate programming techniques to produce routines that carry out the data mining on behalf of the researcher and consequently use the available computer power to extend the research reach of the individual in time and space. Importantly, these advantages are not limited to those who have the programming skills to create those routines in the first place. Today's programming languages enable the development of applications that make creation of programming routines a matter of assembling a sequence of preprogrammed modules, that application itself writing the code that will eventually carry out the job. In much the same way that presentation applications allow the user to assemble a multimedia presentation without need of programming skills to bring together text, images, video and sound, so these data mining applications allow the non-programmer to assemble a sequence of routines that retrieve sequences, query databases, retrieve results and then, potentially, formulate new searches on the basis of responses to earlier queries.

These applications produce ‘web agents’, which are computer programs that search the Internet on behalf of their author for data that the author requires.

Existing systems for automated access to the Internet resources are usually either specialized for a single given task or written as a general purpose tools. By their levels of complexity, these can be grouped in the following categories:

1. Single task, single step web agents work with a single web service only. For example, WebBlast (Ferlanti *et al*., 1999) stores the search requests in the local query and schedules submissions to the server (for example, during the night time). BioQuery (Brundege and Dubay, 2003) periodically searches NCBI database (Wheeler *et al*., 2003) using the previously stored queries. As a rule, such systems automate a submission only. The returned response is analyzed by the user.

2. Multiple task, single step web agent systems are collections of the single task web agents, having shared user interface and shared mechanisms for integrating the new web services. For example, Sewer (Basu, 2001) is a JavaScript based system of web pages containing web forms for accessing various bioinformatics web services. These forms replace the original forms of the corresponding web pages, creating a more effective environment for the user. The Sewer assistance ends after the request is sent. Differently from Sewer, Proteomix (Chikayama *et al*., 2004) needs specialized software on the server side, communicating via a more advanced protocol called SOAP (Simple Object Access Protocol). This tool is more specialized, analyzing the given protein sequence in a variety of ways.

3. Multiple step, fixed workflow systems connect several agents into a workflow. Such systems are frequently used for automated genome annotations. First, some gene prediction service must assemble the sequenced clones and predict the genes for the given DNA sequence. Next, one (or, more often several) other services provide some conclusions about the predicted protein (for example, that there are known proteins to which it is similar, that known functional domains are present, how many transmembrane regions it may contain, and so on). The results are usually stored in a database. Examples of such systems could be Genotator (Harris, 2000), Pedant (Frishman *et al*., 2001) or Fountain (Buerstedde and Prill, 2001).

4. Changeable workflow, fixed agent set systems are used for flexible workflows, where the agents for the workflow are picked from a fixed list. The integration of new services still needs serious programming effort in these cases. Several such systems were suggested quite a long time ago, usually adapting already existing frameworks for the needs of bioinformatics analysis. In this category we could mention Kleisli (Kolatkar *et al*., 1998), GAIA (Bailey *et al*., 1998) or Tambis (Stevens *et al*., 2000).

5. Changeable workflow, extendable agent set systems allows the end user to add new agents without significant programming effort. Depending on the method

by which the new agents are added, this group falls into the following subgroups:

a) Systems in which the originators take pains to create a simple way for adding new agents written by user proficient at programming. Many projects (for example EDITtoTrEMBL) support integration of the user-written code, and this possibility usually remains in more advanced systems like Sight or Taverna.

b) Systems that automatically generate code templates that must be finished by the user with a little additional programming. In this case the system itself usually focuses on handling the communication between agents. A good example of such a skeleton generator is Decaf (Graham *et al.*, 2003).

c) The new agent is generated using a graphical user environment, which is the approach employed by the Sight program discussed in more detail below. In these cases the operator provides all the necessary additional information, but the program writes the code.

d) The web server that provides the web service the user wishes to access may also offer additional web document(s) that can be used to create web agent(s) for the service. This document defines the formats of the possible requests and responses. It is usually written in WSDL, the machine-readable language created for describing web services. The server must also be connected using a specific protocol, different from the protocol used to submit a web form. The most advanced experimental applications like Taverna (Oinn *et al.*, 2004) rely exclusively on this new approach and cannot integrate the ordinary web services. We think that a more useful approach could be to add WSDL support without discarding the ability to create agents for work with the classic web forms.

We have created an application called Sight, which is a Java™-based package that provides a user-friendly interface to generate and connect agents for automatic genomic data mining, allowing the user without programming skills to tailor web agents for his/her individual requirements (Meškauskas *et al.*, 2004). Sight allows the assembly of an arbitrary flowchart-like workflow. Using a Web form the user chooses agents from a built-in library and connects the response data fields of one agent to the request data fields of another. This user-interactive agent generator produces agents that can be connected for sequential tasks using the application. To minimize Internet connections for trivial tasks, Sight incorporates the algorithms for several functions such as pattern searches, protein translations or simple sequence manipulations have been included in the application for local use.

There are several other facilities available in the application, but Sight's built-in web agent generators have never been tested on the fungal-specific Internet resources. Web pages devoted to such resources were designed later than pages devoted to plant or human sequences. As a result, they make intensive use of advanced features like JavaScript language, unusual (often nested) tables, multiple pages per response, and

so on. Such features provide a serious challenge for web agent applications that must still be able to extract a clear data structure from a complicated multi-page server response. The purpose of this work was to test and adapt the Sight system, enabling easier creation of fungal-related workflows. The list of web resources used to tailor Sight to fungal genome servers (Table 1) was taken from the chapter by Moore *et al.* (this volume).

Table 1. The fungus-related Internet resources supported by the most recent version of the Sight web agent application (version 3.0.0).

Organism	URL	Resources
Aspergillus fumigatus	http://www.tigr.org/tdb/e2k1/afu1/	Similarity search (nucleotide sequences only)
Cryptococcus neoformans	http://www.tigr.org/tdb/e2k1/cna1/	Similarity search (protein and nucleotide sequences)
Phaenerochaete chrysosporium	http://genome.jgi-psf.org/whiterot1/whiterot1.home.html	Similarity search (nucleotide sequences only).

2. RESULTS

A Sight web agent is essentially an active flow chart in which each element is a working preprogrammed routine. The user assembles the flow chart according to the task s/he wishes to perform. I will outline here the specific changes that have to be made, either to the program or to its implementation, in order to apply Sight to data mining of fungal genomes.

2.1 Main conceptions of the Sight workflow

Sight architecture has been significantly modified since the program was first briefly described in the literature (Meškauskas *et al.*, 2004), by the addition of loops, confluences (convergences), and so on. These new features will be described below.

2.1.1 Sight agent

The reusable elementary unit (or agent) executes a single remote or local algorithm. For this we need two data structures, defining the submitted request and the received response.

Sight request consists of the multiple named items (fields), each storing a string value. They normally correspond to the fields, checkboxes and other controls in the web form that was used as the initial data to generate the agent. In contrast to the request, the result of the bioinformatical web agent often needs to be an array of records. For example, a similarity search service may return multiple hits to the sequences in the database; a gene prediction program may find multiple genes in a DNA sequence; a program for predicting transmembrane segments may detect multiple transmembrane helixes, etc. Hence, Sight agent response is programmed as an array of records. These records also consist of multiple named fields.

As the request and response format differs for each agent, the agents also contain explanatory data structures defining these formats. For each request or response field they define its type, name and arbitrary comment. The request fields can also have a default value and a list of other possible values.

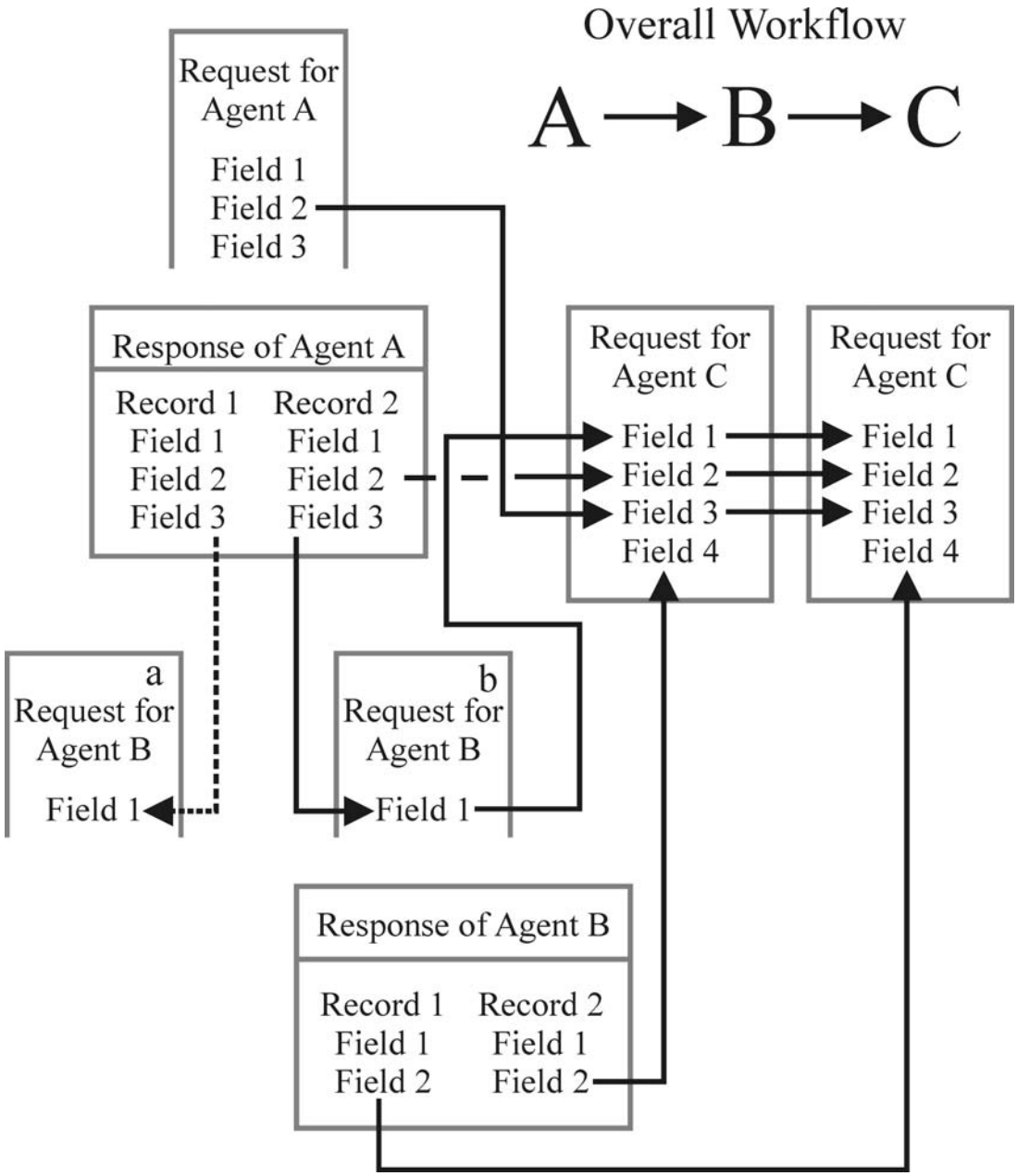

Fig. 1. An illustration of data flow during data type conversion for the case of the simple linear workflow employing three Sight agents A, B and C. The initial request for the agent A consists of 3 fields. This agent returns a request from two records. Each of them also have 3 fields. The request of agent B consists of one field, and the value for this is taken from field 3 in the agent A response (= results) record. As agent A returns two records, two independent requests (**a** and **b**) for agent B will be created (shown as dashed and solid lines). Let us suppose that for one of these two requests agent B returned two records, each having 2 fields. Now, finally, the request is made to agent C consisting of 4 fields that must be filled by various values from the workflow. Field 1 is identical to the field from the agent B request, field 2 is identical to the field 2 from the agent B results record, field 3 takes its value from field 2 in the initial workflow request for agent A, and finally field 4 takes its value from field 2 in the agent B response record. As agent B has returned two records to the request **b**, two requests for agent C will be created for this branch. However as agent B also has another request (**a**), the total number of requests to agent C depends on the number of records in the agent B response to its request **a**. If this response contains, for example, 3 records, the total number of the requests to agent C will be 3 + 2 = 5.

2.1.2 Sight workflow

For any workflow, connection between two agents it is only possible if the result of the master agent can be converted to the request for the slave agent. Some systems require that these two data structures should be identical or (like Decaf) leaves the solution of the problem to the programming user. On the other hand the Sight application generator produces Java™ code to create a slave request from the master response. More exactly, the request is created using the full hierarchy of response and request (the master response, master request, the master of master response, the request that was sent to the master of master and so on) right up to the level of workflow input data (Fig. 1).

The code for such a workflow is generated automatically; the user simply connects the required request and response fields. Fig. 2 illustrates the simple case of a branching, tree-like workflow. As shown in this example, the system can potentially generate a large number of requests, especially for the agents standing lower in the hierarchy. For example, if the master agent has returned 7 records, these will represent 7 requests for the slave agent. If this slave agent has returned, for example, 5 records for each request and has its own slave agent, the number of requests for this slave-of-slave will already be 30. The inbuilt Sight security system limits the number of parallel submissions to the same web service to prevent server overload.

Previous Sight versions supported linear and tree like workflows only. However, the current version (v. 3.0.0), as presented in this manuscript, also supports loops and confluences (Figs 3 & 4). The confluence arises when two or more branches of the tree workflow must join together again to provide data for a shared agent (Fig. 3). Our solution for the type conversions for confluences is to process all possible combinations of the records in the two master agent responses. For example, if the workflow has branched and two slave agents have the shared slave-of-slave agent, and one of these two agents has returned 5 and another 3 records in response, it is possible to combine 15 different requests for the shared slave-of-slave agent.

2.1.2 Loops

Sight 3.0.0 supports circular workflows. Such algorithms are used, for example, in building sequence similarity networks or in reconstruction of metabolic pathways. The loops are realised with a pair of two communicating specialised agents: loop starter and loop closer (Fig. 4). The loop starter just passes all its requests through. When the loop closer receives the request, it communicates the loop starter, initiating the additional “virtual request”. This “virtual request” is processed by the agents between the loop starter and loop closer and may initiate the subsequent new virtual requests. The loop is terminated when one of the agents between the starter and closer returns the empty response (no records) or when the maximal number of iterations is exceeded.

2.1.3 Storing the results

The results of running the workflow must be stored for subsequent viewing or analysis by the user. For systems with a fixed workflow the results are usually stored in the database. However each user-defined workflow usually needs a new database structure. It is difficult to implement a user-friendly interface for accessing these multiple different databases. The older versions of Sight stored the results in the html documents.

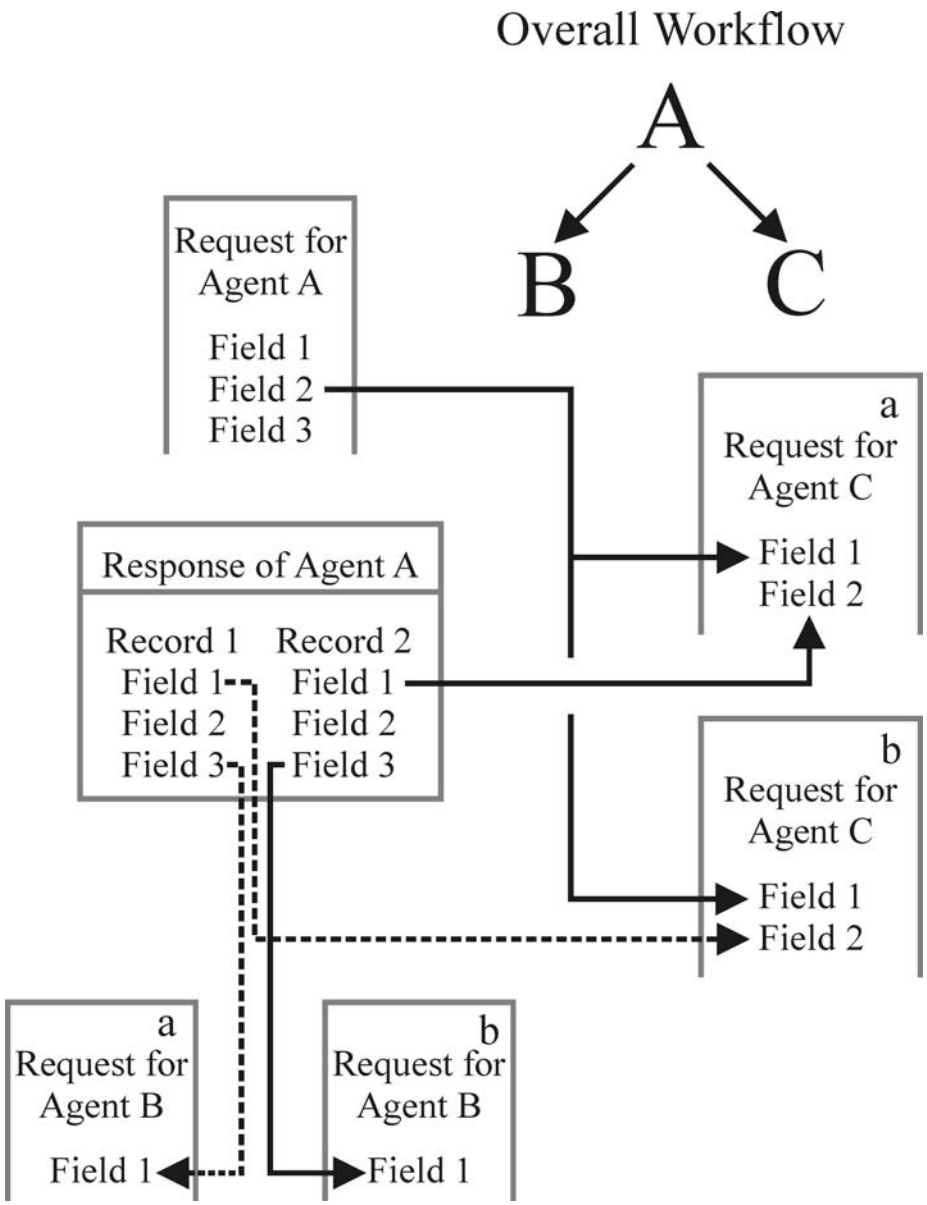

Fig. 2. Illustration of data flow during data type conversion for the case of a branching (tree-like) workflow. Here, agent B requires the value held in field 3 from the master response record. Agent C requires field 1 from the master response record, but it additionally needs the data in field 2 from the master request. As the master A has returned two records in response, both slave agents (B and C) receive the requests (**a** and **b**).

Taverna tried another approach, creating a complicated folder and subfolder structure on the local file system. In the new Sight version we implemented the ability to store the results in the form of network. The agent responsible for storing the network takes the names of the two nodes that must be connected. As the agent receives more and more requests, the number of currently existing nodes and connections increases. The created network can be viewed with the free bioinformatical graph viewer CytoScape (Shannon *et al.*, 2003). Sight also has a specialised group of agents (loggers) that just append the requests to the local files. In this way the interesting information can be logged separately in FASTA or some other format.

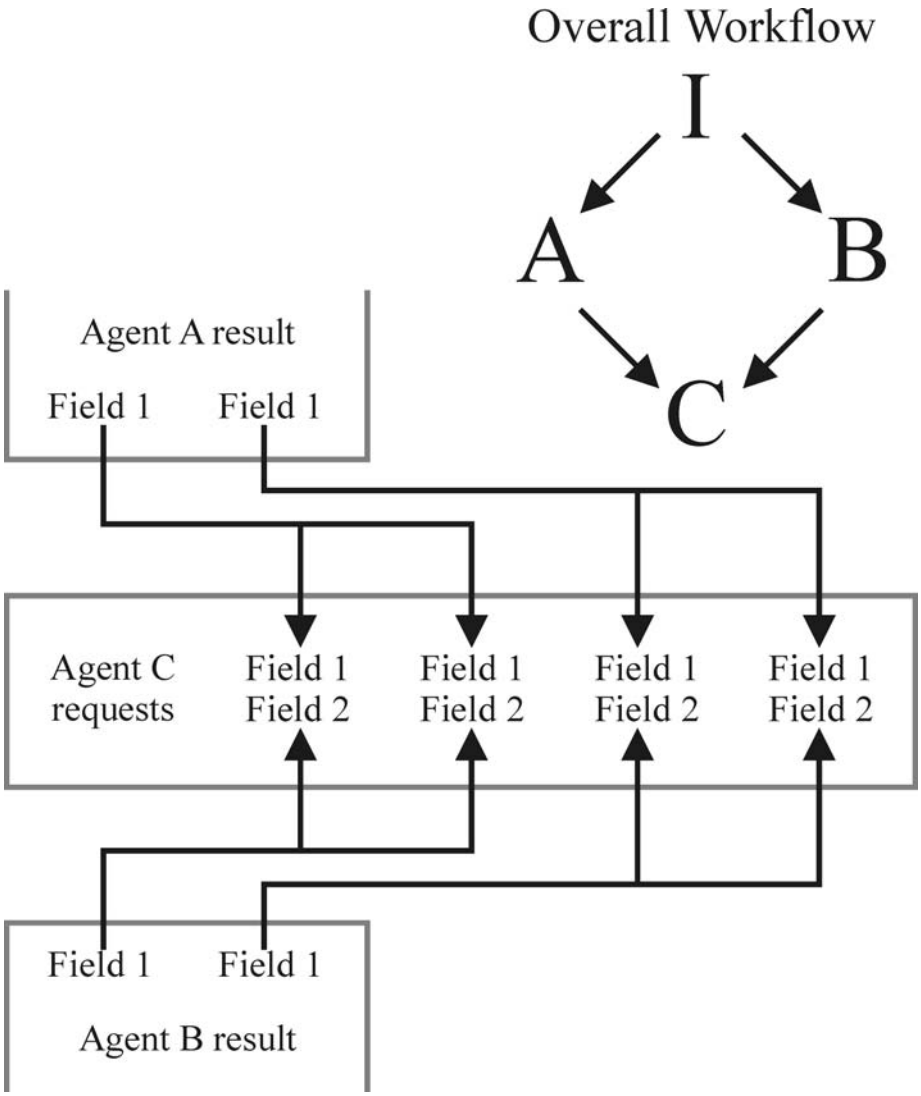

Fig 3. The simple case of confluence. In this workflow, the agent I is a master for two agents A and B. Agent C is a shared slave agent for A and B. If agent I returns a single record result, agents A and B both receive a single request (not shown). However, both A and B results contains two records, and the shared slave needs fields from both master agents, this workflow generates four requests for agent C. Confluences are only supported in the new version of the program Sight 3.0.0 alpha.

2.2. Testing the web form submission module

To create a web agent we must first submit a biologically relevant request; if a similarity search is the main interest then the request must be for a protein or nucleotide sequence that the user expects to be found in the server database. For easier preparation of test requests, the web agent generation system needs some built-in example sequences. As Sight was written as a tool for analysis of the human genome, the program was initially equipped with sample protein, DNA and RNA sequences taken from human sources. Such sequences have no analogues in fungal genomes, so the corresponding similarity searches return no hits. To provide realistic queries, I have extended the built-in example set by adding the RNA and protein sequences of ribonucleotide reductase M2. A similarity search test operation will now find similar sequences in any fungal genomes the user cares to test.

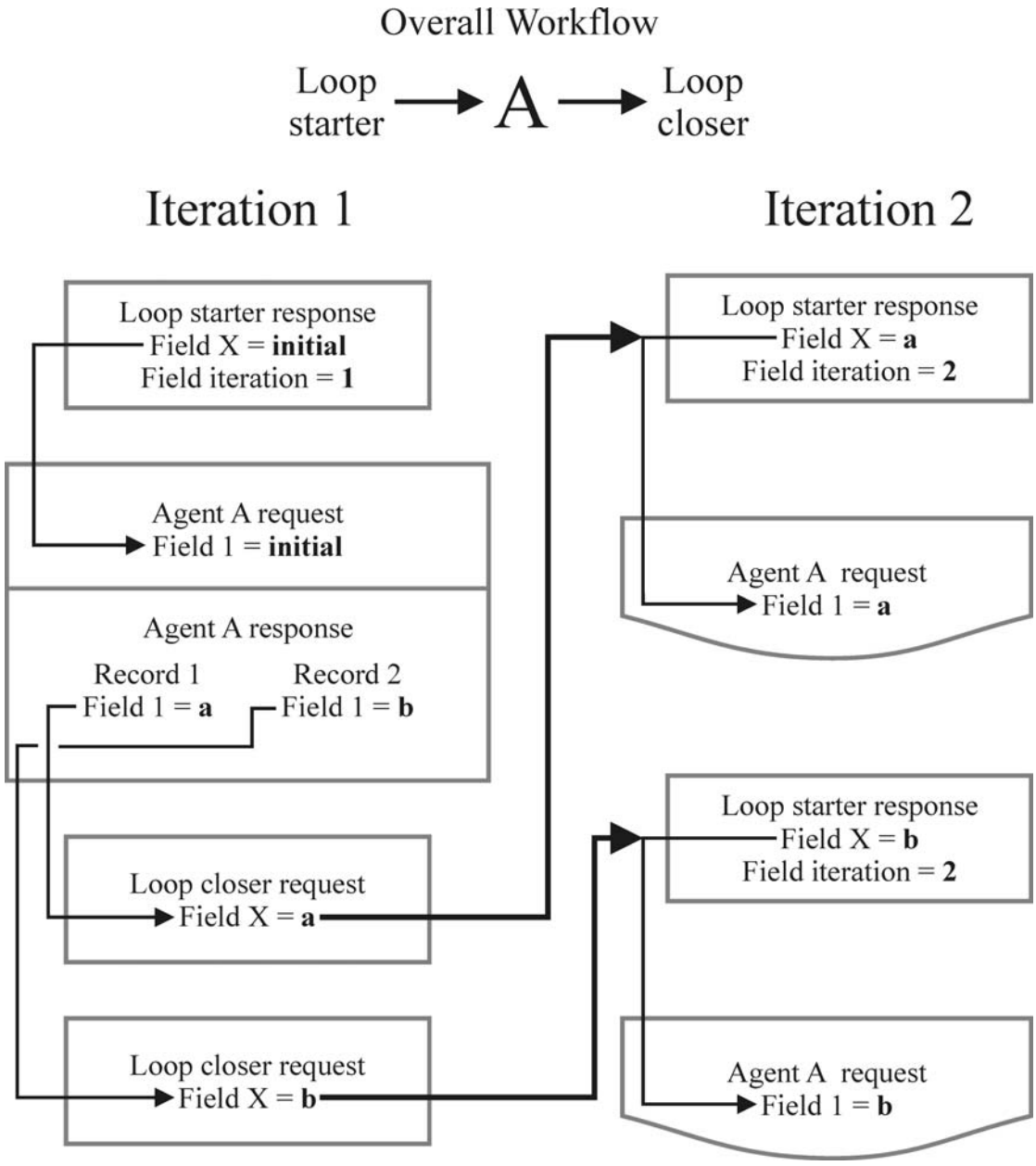

Fig 4. The concept of processing loops in Sight. The figure illustrates a simple loop, where agent A is placed between the loop starter and loop closer. During the first iteration the loop starter send one request to its slave agent A. As agent A returned two records in its result, the loop closer (slave agent for A) receives two requests and during the next iteration produces two virtual requests for the loop starter. The words "**initial**", "**a**" and "**b**" are sample values and illustrate how the structures are converted during iterations. Loop agents can handle up to 3 loop variables (X, Y and Z). Loops are supported in Sight 3.0.0 alpha and higher.

2.3 Testing and extending the redirection system

In the simplest case a web server accepts the web query form the user has completed on-line, performs the requested search, and returns a web page that contains the results of the bioinformatics analysis. The duration of analysis is usually limited by automatic breaks in the connection after several minutes caused either at the client or server side. Recent experience shows that increasing numbers of web services are returning an intermediate page. This page confirms that the request has been accepted and is being dealt with, and contains the hyperlink that must be

followed to retrieve results. If this hyperlink is followed before the server has completed the task, another page is returned suggesting the enquirer might like to be more patient and wait for some further time. Essentially, this strategy allows the server an unlimited amount of time to complete the task. However, most of the fungus-related web services go even further. Instead of returning the complete result, they tend to provide only a synopsis or general description. For example, the result page of a similarity search over the *P. chrysosporium* genome does not contain the score for the similarity search. Such important details can only be retrieved by following additional links from the response page. If the web agent is to extract the user's results effectively, then it must be able to deal automatically with these intermediate pages returned by the database server.

Sight already supported the most trivial cases, but we found that the redirection problem required rather more attention. The first difficulty is that the number of intermediate pages may not be known in advance and they may need different algorithms to find the hyperlink to follow. Another problem is that in some cases the user is expected to follow several links from a single page, which creates a task related to web crawling for the web agent. A new version of Sight has been produced that replaces such a sophisticated agent with a mini-workflow consisting of specialised sub agents that find the hyperlinks the user is expected to follow to get all of the results. Mini-workflows are assembled and tested like the ordinary Sight workflows, but after generation, they join the final Sight application as main agents. Mini-workflows cannot be realised simply as part of the ordinary Sight workflow because a different caching strategy is required. The intermediate hyperlinks that have to be followed must not be cached. Instead, the final result must be cached using the initial request to compute the caching key.

2.4 Testing and extending agent algorithms

It is typical to illustrate the results of searches with graphics. However, the services for which Sight was originally designed report the same information in text form and so until now Sight's web agents did not need to analyze these figures. Unfortunately, fungal genome servers are different. For example, the result of a similarity search of the *P. chrysosporium* genome comprises just one large image illustrating the position(s) of the hit(s) that have been found in the total DNA sequence. To get more information, it is necessary for the user to mouse-click on the image. From the point of view of a web agent, it is necessary to follow those hyperlinks defined in the "hot spots" in the image, but the graphic contains several "hot spots" and only some of them lead to pages with useful additional information. The links that are needed cannot always be recognized from the surrounding data by an automatic routine, but in many cases they can be recognized from the Internet address itself. In the example quoted the link must contain the substring "getAlignment". Hence, we have produced a new Sight version that includes the previously missing feature of selecting data fields by content rather than by context.

Experience with fungal genome servers also showed that the automated table analysis of the original version of Sight also needed serious improvement. Details of search results of the *P. chrysosporium* genome are presented in the form of multiple tables, there being one table per search hit. Instead of working with the single table generated by human genome servers, the fungal web needs to collect results from several very similar tables. Interestingly, these tables themselves are cells of an even larger "supertable".

To solve this problem in Sight, we have now implemented a concept of the table "domain", defining the table position in the nested supertables (Fig. 5).

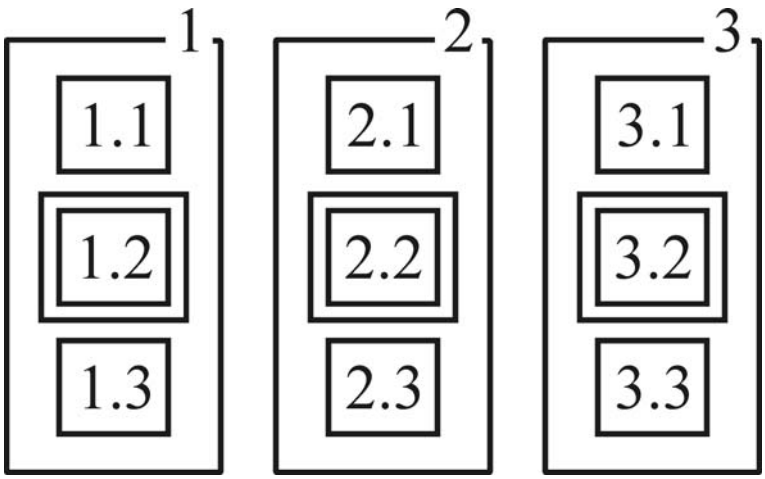

Fig. 5. Illustration of the concept of the table domain on the results web page returned by the genome server in response to a query. In this example, the required data are located in the table in the group of cells shown as 1.2, 2.2 and 3.2 and the web agent identifies these by their content.

2.5. Discussion and Prospects

The version of Sight which has now been tailored to use fungal genome servers will be extremely useful to mycologists involved in data mining. There are still some remaining problems, though. The most challenging among these are servers that return work results to the enquirer by E-mail (for example, the Whitehead Institute servers). It is not difficult to program automated E-mail checking, but the issue that arises is that in complicated workflows there may be several agents issuing requests and there is a problem in deciding which agent in the workflow sent a specific request which is answered by a specific E-mail. For example, if two agents in a workflow submit requests to the same server (say, related searches on different sequences), then how can the E-mail messages be correctly sorted automatically when the server returns them? Evidently, the server response must contain some kind of "submitter name" which the agents can use for sorting. Many servers do provide such support, allowing the user to specify, for example, the sequence header. That header may be used subsequently by the web agents for their own orientation. This is not a trivial problem and, for the moment, this part of the application is not sufficiently reliable, and we are still not ready to release it for wider use.

3. CONCLUSION

During attempts to generate web agents for fungus-related Internet resources (Table 1) it was recognized that some of those resources use new methods of representing the information they report. In some cases it was not possible obtain search details with the previously available version of Sight (v. 2.1.2), and some servers returned multiple intermediate pages leading towards their response which created difficulties for automated recovery of results. Despite these problems, it was possible to use Sight to create web agents that are able to automate searches of fungal genomes. The previous version of the application was adapted with a little additional programming,

creating a new version for which these features of the fungal genome servers do not represent a problem. The new version of Sight (v. 3.0.0) that is tailored to servers carrying fungal databases is freely available for download from the project website at these URLs:

http://bioinformatics.org/jSight/ and http://jsight.sourceforge.net/index_SF.htm.

Acknowledgement: I thank Dr David Moore (School of Biological Sciences, University of Manchester) for making the original suggestion that I should test the Sight application on fungal resources and implement the adaptations that were needed.

REFERENCES

Bailey LC, Fischer S, Schug J, Crabtree J, Gibson M and Overton GC (1998). GAIA: framework annotation of genomic sequence. Genome Research 8:234-250.

Basu MK (2001). SeWeR: a customizable and integrated dynamic HTML interface to bioinformatics services. Bioinformatics 17: 577-578.

Buerstedde JM and Prill F (2001). FOUNTAIN: A JAVA open-source package to assist large sequencing projects. BMC.Bioinformatics: 2, 6-7.

Brundege JM and Dubay C (2003). BioQuery: an object framework for building queries to biomedical databases. Bioinformatics 19:901–902.

Chikayama E, Kurotani A, Kuroda Y and Yokoyama S (2004). ProteoMix: an integrated and flexible system for interactively analyzing large numbers of protein sequences. Bioinformatics: (in press).

Ferlanti ES, Ryan JF, Makalowska I and Baxevanis AD (1999). WebBLAST 2.0: an integrated solution for organizing and analyzing sequence data. Bioinformatics 15:422-423.

Frishman D, Albermann K, Hani J, Heumann K, Metanomski A, Zollner A and Mewes HW (2001). Functional and structural genomics using PEDANT Bioinformatics 17:44-57.

Graham J, Decker K.S and Mersic M (2003). Decaf - a flexible multi-agent system architecture Autonomous Agents and Multi-Agent Systems. 7(1): 7-27.

Harris NL (2000). Annotating sequence data using Genotator. Molecular.Biotechnology 16:221-232.

Kolatkar PR, Sakharkar MK, Tse CR, Kiong BK, Wong L, Tan TW and Subbiah S (1998). Development of software tools at BioInformatics Centre (BIC) at the National University of Singapore (NUS). Pacific Symposium on Biocomputing 735-746.

Meškauskas A, Lehmann-Horn F and Jurkat-Rott K (2004). Sight: automating genomic data-mining without programming skills. Bioinformatics 20:1718-1720.

Moller S, Lesser U, Fleischmann W and Apweiler R. (1999). EDITtoTrEMBL: a distributed approach to high-quality automated protein sequence annotation. Bioinformatics 15: 219-227.

Oinn T, Addis M, Ferris J, Marvin D, Greenwood M, Carver T, Pocock MR, Wipat A and Li P (2004). Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics in press; advance access doi: 10.1093/bioinformatics/bth361.

Shannon P, Markiel A, Ozier O, Baliga NS, Wang JT, Ramage D, Amin N, Schwikowski B and Ideker T (2003). Cytoscape: a software environment for integrated models of biomolecular interaction networks. Genome Res. 13: 2498-2504.

Stevens R, Baker P, Bechhofer, S, Ng G, Jacoby A, Paton NW, Goble CA and Brass A (2000). TAMBIS: transparent access to multiple bioinformatics information sources. Bioinformatics 16:184-185.

Wheeler DL, Church DM, Federhen S, Lash AE, Madden TL, Pontius JU, Schuler GD, Schriml LM, Sequeira E, Tatusova TA and Wagner L (2003). Database resources of the National Center for Biotechnology. Nucleic Acids Res. 31: 28-33.