

BIOINFORMANTS: BIOLogical, INFORMAtional ageNTS on the Internet

Jacinto Dávila¹, Jose López^{1,2}, and Almathely Vivas²
jacinto@ula.ve, jlopez@unet.edu.ve, alma@unet.edu.ve

¹ Centro de Investigación y Proyectos en Simulación y Modelos
Universidad de Los Andes *
Mérida - Venezuela

² Universidad Nacional Experimental del Táchira
San Cristóbal - Venezuela

Abstract. We define a bioinformant as a software agent with specific abilities to serve as an assistant for a biologist. This paper describes the design guidelines and the first implementation of BIOINFORMANTS: a platform for scientific computing on the web. The goal of this project is to develop a computational, web-oriented, software platform to recover and analyze genetic information, (re)using existing software tools for biologists. We aim to a light and intelligent, web application integrating functions and services provided by programs like Phylip [12] and CLUSTAL-W, from the biologist's toolbox, with newer platforms to process biodata, such a PROGOL.

Keywords: Agents, Web Application, Java.

1 Introduction

This paper describes the design guidelines and the first implementation of BIOINFORMANTS: a platform for scientific computing on the web. The goal of this project is to develop a computational, web-oriented, software platform to recover and analyze genetic information, (re)using existing software tools for biologists.

Specifically, we aim to a light and intelligent, web application integrating functions and services provided by programs like Phylip [12] and CLUSTAL-W, from the biologist's toolbox, with newer platforms to process biodata, such a PROGOL. We are adding "intelligence" to this application by means of software agents serving as work assistants, tutors and data-miners. The technologies adding value to the existing tools are 1) Agents in logic programming [6], [17], 2) Inductive Logic Programming and 3) Java Web applications [1].

Phylips [9] is a well-known set of programs to compute DNA sequence similarity and draw dendograms from it. The input to Phylips is a file with distances and relations between DNA sequences, which could be produced by other products such as CLUSTAL-W [2]. So, this integrating effort has contemplated the inclusion of access to Clustaw-W and to sources of raw data such as GenBank

* Fonacit, Venezuela, project number S1-2000000819

[3]. In this sense, our web-oriented development is an achievement, as most of this data providing services are already active on the web.

We are offering the programs that constitute the interface and the web application itself as an easily deployable software, to be downloaded from the web. Notice, however, that the core of the system is the set of server-sided programs, servlets, to integrate pre-existing applications into the web environment. These servlets required the deployment of their supporting software (the TOMCAT container) which is, by the way, freely available as well.

In this paper, we first describe, in section 2 the general architecture of the BIOINFORMANTS. In section 3, we explain how that architecture is being implemented with the Java web applications, by going through a few use-cases. In section 4 we explain our approach to security for these applications. In section 5, we define the agent component and present its current implementation, adapted to serve as a (very elemental) tutor. Finally, in section 6, we explain our plan for deployment and user-guided improvement.

2 The architecture of BIOINFORMANTS

We define a bioinformant as a software agent with specific abilities to serve as an assistant for a biologist. Bioinformants will be helping scientists, we expect, to store, process and analyze their data, based on knowledge specific to the domain, to the application and to the user.

To develop Bioinformants, we are following a progressive approach. We aim to build a multi-agent system, as those known in Artificial Intelligence [18]. But we started with the integration of existing tools into an usable unifying application on the web. We then “inserted” the agents in the environment provided by that application and provided a number of mechanisms for users and agents to interact.

The top-level description of the system architecture is illustrated in figure 1 and it is as follows: There is a UNIX server-machine, running applications such a phylips. The machine is also a web-server, extended to support web-applications (with tools such as the JAVA servlets platform discussed in the following section). There is a web application running on the server and launching pseudoterminals to allow the users to run their programs as in a remote shell, but also spawning bioinformant agents to assist users over the web. There is, of course, a web-page, which can be downloaded into any Internet client with a web browser. An applet, in that web-page, is the user interface to bioinformants.

As a first approximation, bioinformant agents are very simple subprocesses. They are launched (spawned) by the bioinformant-server simultaneously with the spawning of a new pseudo-terminal for the user. A bioinformant will take messages explicitly addressed to it, expressed in “controlled” form of natural language, and process them according to its own set of inner goals, rules and beliefs. At the same time, the user is free to send messages, typically commands to a UNIX shell, asking the operating system for their execution and collecting the output on the interface, which is sent back to the user over the web.

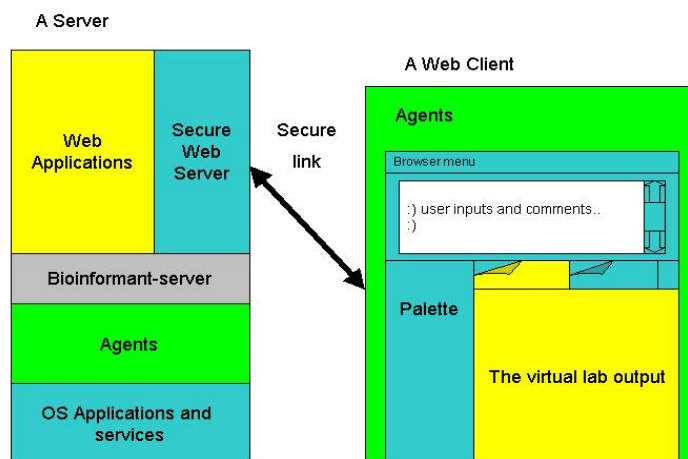


Fig. 1. The architecture of BIOINFORMANTS

The Bioinformants' design is founded on the following hypothesis about this domain of applications:

1. To integrate scientific tools, one has to respect established usages and procure economy (i.e. WWW and Open Software).
2. Scientist do not want to spend their time re-training on software tools, configuring them or looking for extra computing resources to support them (i.e. light web interface, server/client architecture, legacy bio-software, Linux and PCs).

Thus, bioinformants is built on those tools, platforms and constraints. We have also started to elaborate on the simple model of figure 1 (after completing it with other services, such as security) to allow for:

1. A rich interaction language for the final user. That is, not only command in unix-like formats, but also more natural-language-looking sentences. It is important, however, to remember that these sentences are in a **controlled** form of natural language, to avoid the complexity of the general case. So, there is no claim of having a natural language interface, whatsoever. We do have an interface able to answer partially predefined queries. And this is done locally (in the interface applet).
2. A reasoning process mediating between inputs from the user and outputs to the user. That is, we have the agent reasoning about what it is being requested, what it is being done by the applications and what it is being answered to the user. Current examples are very elemental, but they suffice to show the possibilities.

3. Knowledge engineering about applications for biologist and genetic engineers. This has led us to elicit knowledge that could be encoded into the assistant agent, as shown in section 5.
4. Different types of agents in the systems. To a certain extend, the interface applet can be regarded as an agent that can “understand” messages from the user and discriminate them into commands and data (for the terminal), queries (for itself) and questions for the agent.

We have already implemented the “first approximation” and this is described in the following section.

3 The implementation as JAVA Servlets in a Web application

As we said, we have implemented the first approximation to Bioinformants. The base platform is JAVA and the servlets [1]. By extending a http server with an implementation of the Java Servlets specification, code-named TOMCAT [4], we were able to deploy a piece of JAVA code that serves non-web applications installed in the same machine, through the web. We called it the BioTerminal. Figure 2 shows how we have instantiated the architecture in 1, for this first approximation.

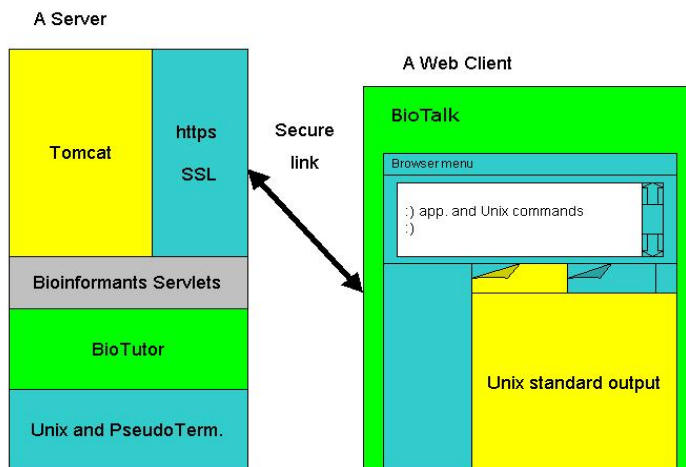


Fig. 2. The current state of bioinformants

The current implementation is, perhaps, better explained by a diagram with a use-case. Figure 3 shows a diagram in which every polygon represents an object. Arrow-like polygons (unlike the boxes) also represent threads or separated

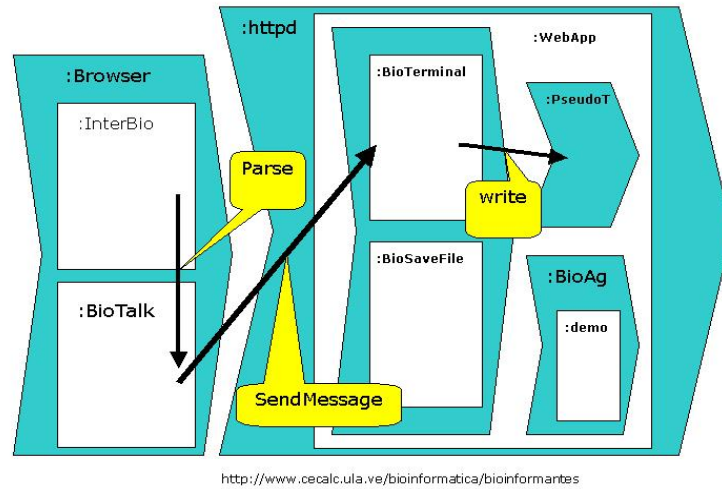


Fig. 3. use case one: sending a command to be executed

subprocesses. The name above each polygon refers to a class (although in some cases is only a simplification, like in **:httpd**).

This diagram says that there are two separated computing groups. One within the browser (which includes the objects **:InterBio** and **:BioTalk**) and the other one on the server, within the TOMCAT web container. Notice that there are, at least, three threads on this group, with the pseudo-terminal and the agent occupying their owns. Communication internally to each group is by traditional object message exchange and also via calls to the operating system. Communication between the two groups is based on https.

We wanted an operational prototype in a short time, so that could start offering services to a selected community of scientist (like the use of certain UNIX applications) on the web and get some feedback to focus the developments therein. Figure 4 shows a picture of the interface already functional. The system can be tested at <http://www.cecalc.ula.ve/bioinformatica/bioinformantes/>.

However, no web application is usable without a solution for secure access and data handling.

4 Security

The JAVA servlet platform does include a number of facilities to deal with secure access over the web. We chose a solution that uses a small record of users kept in the server. This is the simplest solution and it is documented in [4].

It is important to note, however, that the main element of security is not the authentication mechanism, but the mechanism that encrypts all the data that it



Fig. 4. Bioinformants interface

being manipulated by the user and by the web application. This is why we are delivering information using SSL. Ours is a secure web server.

Non-the-less, the more interesting component of the Bioinformant platform is the agent technology.

5 Agents

An agent in Artificial Intelligence is a device capable of sensing its environment and act upon it [18]. This definition is of little use, however, to characterize what is being called an agent or, moreover, an intelligent agent in modern computer science. An agent senses and acts, but it also performs some form of reasoning to relate perceptions to actions and to its own agenda. An agent, it is widely accepted, must be seen as an *intentional system*: a system with intentions and informational attitudes such as goals and beliefs. This human-like conception of an agent is useful because it allows for the description of an entity that it is autonomous in pursuing the goals for which it has been programmed. It is been argued that, just like objects are a powerful abstraction to support the development of computer systems, agents are very suitable for the development of complex computer systems. One of the key elements in this argument is that the developers of agent-based systems can concentrate on the high-level description of the “know-how” and the goals for the agents, leaving to them the final decisions about when exactly they must do what.

Thus, we decided to draw from these recent developments in Artificial Intelligence to build a bioinformatic system. We chose an particular agent technology

in which we had experience and which also leave open several line of extension and enrichment: logic-programming agents [7, 8].

We are building on a proposal by Prof. Bob Kowalski which, basically, prescribed the use of logic programs to specify agents that are both reactive and rational [13], [14]. We had developed that proposal into a complete specification by including the specification of a proof procedure that is used as the reasoning mechanism of the agent [6]. This specification is completely translatable into logic programs and is, therefore, an *executable specification*, in the sense discussed in [19]. Table 1 shows the main formulae of the specification we have code-named GLORIA³. The reason for this is that we expect to try different specifications in the future.

GLORIA's cycle is specified, in first-order logic, in table 1 and it is described in detail in [6]. The intended reading of this specifications is summarized as follows:

- The *cycle* predicate, [**GLOCYC**], describes a process by which the agent's internal state changes, while the agent assimilates inputs from and posts outputs to the environment, after time-segments devoted to reasoning. The reasoning process is modelled by the *demo* predicate and the sensing-acting mechanism is modelled by the *act* predicate.
- The act predicate ((**GLOACT**) and [**GLOEXE**]) explains how the agent selects all its planned actions that should be executed at a particular time, and post them, in parallel, to environment. The environment, according to the try predicate, [**TRY**], will admit these actions and try their simultaneous execution, answering back to the agent with the outcome. This feedback has the form of observational facts, that can be added to the agent's knowledge base for further processing.

We call *influences* the things that are being posted to the environment, as they have being called in the multi-agent theory after [10].

Observe the arguments for the demo predicate. This predicate reduces goals to new goals by considering its knowledge base. “demo” is, precisely, the embodiment of the definition of “believes”, the relationship between an agent and its beliefs. An agent believes what it can DEMOnstrate.

This explains the first three arguments of the demo predicate. The fourth argument is an important device to count the amount of resources or the time available for reasoning. At each cycle, the agent reasons for a bounded amount of time and it is so prevented for jumping into infinite regress or total alienation from its environment. This is a legitimate resource in the specification language that allows us to encode an important dimension of the bounded rationality found in realistic agents. A full description of the demo predicate is in [14] and [6].

An important challenge in Bioinformants is to provide a suitable encoding of knowledge for these artificial agents to work in bioinformatic applications.

³ GLORIA stands for a **G**eneral-purpose, **L**ogic-based, **O**pen, **R**eactive and **I**ntelligent Agent.

GLORIA's cycle	
$cycle(KB, Goals, T)$ $\leftarrow demo(KB, Goals, Goals', R)$ $\wedge R \leq n$ $\wedge act(KB, Goals', Goals'', T + R)$ $\wedge cycle(KB, Goals'', T + R + 1)$	[GLOCYC]
$act(KB, Goals, Goals', T_a)$ $\leftarrow Goals \equiv PreferredPlan \vee AltGoals$ $\wedge executables(PreferredPlan, T_a, TheseActions)$ $\wedge try(TheseActions, T_a, Feedback)$ $\wedge assimilate(Feedback, Goals, Goals')$	[GLOACT]
$executables(Intentions, T_a, NextActs)$ $\leftarrow \forall A, T (do(A, T) is_in Intentions$ $\quad \wedge consistent((T = T_a) \wedge Intentions)$ $\quad \leftrightarrow do(A, T_a) is_in NextActs)$	[GLOEXE]
$assimilate(Inputs, InGoals, OutGoals)$ $\leftarrow \forall A, T, T' (action(A, T, succeed) is_in Inputs$ $\quad \wedge do(A, T') is_in InGoals$ $\quad \rightarrow do(A, T) is_in NGoal)$ $\wedge \forall A, T, T' (action(A, T, fails) is_in Inputs$ $\quad \wedge do(A, T') is_in InGoals$ $\quad \rightarrow (false \leftarrow do(A, T)) is_in NGoal)$ $\wedge \forall P, T (obs(P, T) is_in Inputs$ $\quad \rightarrow obs(P, T) is_in NGoal)$ $\wedge \forall Atom (Atom is_in NGoal$ $\quad \rightarrow Atom is_in Inputs$ $\wedge OutGoals \equiv NGoal \wedge InGoals$	[GLOASSI]
$A is_in B \leftarrow B \equiv A \wedge Rest$	[GLOISN]
$try(Output, T, Feedback) \leftarrow tested\ by\ the\ environment..$	[TRY]

Table 1. GLORIA's specification in logic

5.1 Biotutor: an agent customized for bioinformatics

We want to replace the trivial command-processing scheme, used in the first approximation to Bioinformants, with a GLORIA agent, aiming to more elaborated and intelligent behaviour by the server.

```
executable(open_class).          executable(show_page_1).
executable(close_class).        executable(answer_question).
executable(do_method).          executable(do_yes).
executable(answer_question).

observable(biotutor_requested).  observable(class_opened).
observable(seen_page_1).         observable(user_continue).
observable(pending_queries).     observable(want_a_demo).
observable(method_running).      observable(question_asked).

% Integrity constraints.
if biotutor_requested, not(class_opened) then open_class.
if class_opened, not(seen_something) then show_page_1.
if class_opened, seen_page_1, not(pending_queries) then close_class.
if seen_page_1, want_a_demo then do_method.
if user_continue, method_running then do_yes.
if question_asked then answer_question.

% Definitions
to seen_something do seen_page_1.
to seen_something do
user_continue.
```

We also aim to combine the logic-based representation this agent will use, with another logic-based account of knowledge obtained by datamining on bio-data [15].

6 Further work

We are working in several components of the system, both at the architectural level and at the application level. One of the later is an extension to Philyp to provide for new models for genetic analysis.

We also want to produce a well-documented set of experiment on biodatamining. We want to pose a collection of biodata to mining, using a number of different mining techniques and, in particular, tools that allows for detailed explanation of the mining results ([11], [15], [16] and [5]) and tools that allows for human interaction. We expect that human accountability of the mining process and a “human-in-the-loop” mechanism may be particularly productive and attractive for the biologists.

7 Acknowledgements

The authors want to thank the support given to this work by the Venezuelan Funding Agency, Fonacit, under project number S1-2000000819, and by the National Center for Scientific Computing: CecalULA.

References

1. <http://java.sun.com>.
2. <http://www.csc.fi/molbio/progs/clustalw/clustalw.html>.
3. <http://www.ncbi.nlm.nih.gov/Genbank/GenbankSearch.html>.
4. <http://jakarta.apache.org/tomcat/tomcat-4.0-doc/index.html>.
5. Page David, *Ilp (inductive logic programming): Just do it*, Proceedings of Computational Logic CL2000. Lecture Notes in Artificial Intelligence -1861. Springer (2000).
6. Jacinto Dávila, *Agents in logic programming*, Ph.D. thesis, Imperial College, London, 1997.
7. ———, *Openlog: A logic programming language based on abduction*, Lecture Notes in Computer Science (Proceedings of PDP'99. Pars,France) Gopalan, Nadathur (Ed.). Springer. ISBN 3-540-66540-4. **1702** (1999).
8. ———, *Actilog: An agent activation language*, Practical Aspects of Declarative Languages (New Orleans, LA, USA) (V. Dahl and P. Wadler, eds.), LNCS, no. 2562, PADL 2003, Springer, January 2003, p. 194 ff.
9. J. Felsenstein, *Phylogenies from molecular sequences*, Inference and Reliability Annual Reviews of Genetics. **22** (1988), 521–65.
10. Jacques Ferber and Jean-Pierre Müller, *Influences and reaction: a model of situated multiagent systems*, ICMAS-96, 1996, pp. 72–79.
11. Peter Flach, *Knowledge representation for inductive logic programming*, CL2000, Imperial College, London, 2000.
12. <http://evolution.genetics.washington.edu/phylip/credits.html>.
13. Robert Kowalski, *Using metalogic to reconcile reactive with rational agents*, MetaLogics and Logic Programming (K. Apt and F. Turini, eds.), MIT Press, 1995, Also at <http://www-lp.doc.ic.ac.uk/UserPages/staff/rak/recon-abst.html>.
14. Robert Kowalski and Fariba Sadri, *Towards a unified agent architecture that combine rationality with reactivity*, LID'96 Workshop on Logic in Databases (San Miniato, Italy) (Dino Pedreschi and Carlos Zaniolo, eds.), July 1996, Also at <http://www-lp.doc.ic.ac.uk/UserPages/staff/rak.html>.
15. Stephen Muggleton, *Scientific discovery using inductive logic programming*, Communications of the ACM **42** (1999), no. 11.
16. Stephen Muggleton and Wray Buntine, *Machine invention of first-order predicates by inverting resolution*, Proceedings of the 5th International Workshop on Machine Learning. Morgan Kaufmann (1988), 339–351.
17. Elizabeth Pollitzer and Dávila Jacinto, *Application of agent architecture to modelling human-computer interactions*, Proceedings of the ESSLLI'97 Symposium on Logical Approaches to Agent Modelling and Design. Aix-in-Provence, Francia (1997).
18. Stuart Russell and Peter Norvig, *Artificial intelligence: a modern approach*, Prentice Hall, 1995.
19. Michael Wooldridge and P Ciancarini, *Agent-oriented software engineering: The state of the art*, Springer-Verlag Lecture Notes in AI. **1957** (2001).