BioAgent: A Mobile Agent System for Bioscientists

Emanuela Merelli and Rosario Culmone
Dept. of Mathematics and Computer Science, University of Camerino
via Madonna delle Carceri, I-62032 Camerino, Italy
email:{emanuela.merelli, rosario.culmone}@unicam.it

Leonardo Mariani
DISCO - Università degli Studi di Milano Bicocca
via Bicocca degli Arcimboldi, 8, I-20126 Milano, Italy
email:{mariani}@disco.unimib.it

Abstract

The postgenomic era is characterized by the large amount of data available from sequencing experiments. Each scientific institute works on different projects; therefore, data increases rapidly as do the difficulties in managing and sharing information among all potential users. Browsing and searching are expensive activities because extensive time is required to locate the appropriate information. To support the daily work of a bioscientist, we propose to use mobile agent, a computational unit capable of migrating to different places from any location. We then introduce BioAgent, a mobile agent-based system aiming at: 1) complete decentralization of local tasks processing that presently characterizes a workflow of an experiment; 2) reduction of network traffic, thereby freeing researchers from network faults during the experiment, and 3) freeing bioscientists from the need for continuous connection to a laptop. The system is simple, modular, easy to use and implemented in Java.

Keywords

mobile agent system; service agents; user agents; workflow; computational biology.

1 Introduction

At the present, a common problem among biologist researchers, physicians and lab-technicians is the sharing of high volume data which is often raw, heterogeneous and distributed over the Web, and which changes continuously. In the last few years the problem has been partially solved with the advent of huge common databases used both to publish experimental results and to carry out experiments. The great expectations for genome diagnosis, leads to anticipate an ever-increasing number of potential users and data repositories, and consequently an increase in network traffic. Since an experiment could consist of a sequence of steps each of which may involve different data repositories and different experimental techniques placed into remote sites distributed all over the world, a computational biologist researcher usually directly manages and controls the experiment flow by interacting step-by-step with the service's interface of the remote location. User-friendly web interfaces and search engines are the most widely used information gathering tools. However, often the amount of data to be processed is prohibitive for complex interactions. The lack of bandwidth is one of the main limitations on user activities. The internet infrastructure is unable to efficiently transmit and receive so much data. Furthermore, interesting experiments that might contain relevant information may have been carried out in locations which have escaped notice. Certainly it is impossible to search among all information sources, but it is possible to perform thorough and exhaustive automatic searches.

This situation can be improved by using mobile agent-based systems. A mobile agent is a computational unit capable of migrating to different places from any location. Mobile agents diminish network traffic caused by data transfer, thus allowing transmittal of the program to data resources. An agent can interact remotely with its user, behaving in a pro-active, adaptive and reactive way. Agents do not require the user's presence and can therefore be assigned a task to be sent over the web, after which transmittal the connection can be severed. Agents working off-line can provide us with results the next time we login. Agent technology is also related to the artificial intelligence field; therefore, intelligent information extraction, information integration, planning activity, pro-active and reactive behavior, and other typical features are present in multi-agent systems and in mobile agent system.

Define an experiment as a workflow[12], that is a coordinated execution of multiple tasks or activities, where one task consists in visiting a set of data repositories distributed over multiple locations and for each location extracting a certain information, if any. Suppose to associate an agent to the given task, the agent travels among multiple locations, called places, and at each location performs its mission, and at the end of the trip, an information integration procedure takes place before the answer is deployed to the user.

In this work, we propose BioAgent, a mobile agent system suitable to support bioscientists during the process of genome analysis and annotation. BioAgent supports aims at: 1) complete decentralization of local tasks processing that presently characterizes a workflow of an experiment; 2) reduction of network traffic, thereby freeing researchers from network faults during the experiment, and 3) freeing bioscientists from the need for continuous connection to a laptop.

BioAgent is a 4-layered system: the *Core* layer provides basic features of any mobile agent platform; the *Service agents* layer consists of the set of services available in a single place; the *Bio agents* layer hosts and manages the set of mobile user agents. And the *Workflow* layer provides a definition language, called BioAgent-ℓ which, as we discuss in the paper, is appropriate and provides ease of handing for the definition of a bioscientist workflow in terms of coordinated execution of multiple tasks or activities.

Even if many mobile agent systems have been developed in both industry and research, [19, 1, 10, 14], we propose a new one starting from the application point of view – biological domain – and from the concept of simplicity. To that purpose, the *Core* layer of the system has been based on Macondo platform [2] and is completely implemented in Java[13]. The system's simplicity is given by the unique class *Agent* which gives the basic features to each mobile agent, and by two its extension: *UserAgent*, the agent prototype that a user can be refer to perform a task, and *ServiceAgent*, the agent prototype that a platform can host to offer a service. For each of the above agent is defined a proper *SecurityManager*, which forbids any user agent to access both local and remote system's resources, and which controls the execution of the *ServiceAgent* during the the access to system's resources. Furthermore, we have developed three services: HTMLWrapper[18], Ontology[4], WebInterface. The workflow management is under development.

In this paper we introduce a mobile agent system for supporting the complete decentralization of local tasks processing that presently characterizes a workflow of an experiment in the postgenomic era. Such a system, that we called BioAgent is based on simplicity and ease to use. Preliminary results have proved that BioAgent can represent a basis to create a community of users working in the biological application domain.

This paper is organized as follows: Section 2 presents BioAgent architecture; Section 3 gives an overview of the Bioagent implementation; Section 4 presents the BioAgent abstract model and outlines a simple application; Section 5 concludes the paper.

2 BioAgent Architecture

The BioAgent architecture is a layered software architecture (see Fig.1) composed of four different layers, each one providing features at increasing level of abstraction: the *Core* layer, the *Service agents* layer, the *BioAgents* – i.e., user agents layer – and the *workflow* definition environment. The two classes, user and service agents respectively, are distinguished by the differing credentials permissions assigned to each at the time of creation. Only service agents can access local resources; in order to avoid improper and dangerous accesses, user agents can access local resources only by interacting with the local service agents. Thus in any place, an agent is either a user agent or a service agent. The BioAgent framework is based on the definition of service agents as managers of resources. The operating system is defined as a collection of programs designed to manage the system's resources [11] Thus the BioAgent system is a collection of service agents designed to manage the place's resources. The layered software architecture presents several advantages [17]: facilitated partitioning of complex problems into several steps to be performed incrementally; ease of update and reuse of the architecture components. The resulting distributed system for bioscientists is a network of interconnected places, each offering one or more biological services as access to local data repository; the use of local available experimental techniques, etc..

2.1 Core Layer

The *Core* layer contains a set of functions for: security management, resource management, mobility

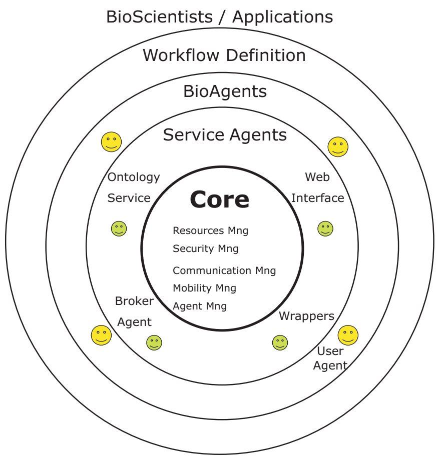

Figure 1: BioAgent Software Architecture

management, communication management and agent management.

Security Management

The security component intercepts any agent's request and verifies the caller's credentials for execution of the requested operation. This control applied to incoming user agents prevents improper use of the system by malicious agents. Potentially dangerous agents or those lacking proper credentials are rejected.

Resource Management

The resource management component provides all basic primitives with access stored data. Credentials verification for access to local resources is left to the security manager. Any service can also implement additional features on data control or on limitation of access; in such instances, control is exerted by the service agents layer. It is possible to ensure that the agent never gains direct control or direct access to a resource, and can only collect responses to its request. If an agent passes the security manager control, the permission check, and any other permission policies encapsulated in the service it intends to use, the resource manager cannot refuse any operation because the negation of a service, in our system, is not conceptually located in the resource manager, and also because the resource manager is not aware of the user agent's identity.

Mobility Management

Mobility supports agent migration. Our system supports weak mobility; consequently, once a migration is performed, the agent's execution state is cancelled. The use of weak mobility is not a problem because the agent preserves its internal state and uses this information to continue its work once a migration has been performed.

Agent Management

The agent creation and cloning manager supports user and service agents for the creation of other similar agents. There are two ways to create agents. In the first, an agent A can clone itself creating an agent B; the new agent B becomes an identical copy of the creator A, except for the unique identifier. In the second instance, another agent is created without using its own internal state as initializing data. The initializing data is provided explicitly by the creator according to the creator's specifications.

Agent creation is invoked by an agent, but monitored by the agent manager at core layer to avoid the presence of too many such agents. Uncontrolled creation of agents could lead to system collapse due to agent overcrowding. Leaving agent creation to the core affords better management policies, for example limitation of the number of agents created by each agent or limitation of the type of agents simultaneously present on one place.

Core primitives allow any agent to communicate, migrate, create other agents or clone itself.

Using only core primitives, an agent cannot access resources. This is appropriate because an agent directly accessing resources would violate security policies. Services guarantees the right level of abstraction and provide limitations on resource access.

Communication Management

A system's agent can communicate with every other agent. The platform's architecture allows messages to be sent to other agents (using their identifier ID), so that, in theory, it is possible to send messages to anyone.

The communication paradigm we have chosen follows a hierarchical structure: each agent may learn the ID of its father agent (its creator) and its sons agents (agents created). This model of communication lends itself well to problem decomposition: to solve a "problem" we can create a set of agents solving "sub-problems". Following the "divide et impera" principle, the main problem is solved by collecting and integrating partial results.

This choice is supported by the hypothesis that our multi- agent system is not open, so that unpredictable communication with unknown agents is precluded.

All aware interactions take place between sons and are coordinated by fathers.

One minor exception concerns access to services, the only case in which, the agent sends a message to another agent that is not a father or son.

2.2 Service Agents Layer

Above the core layer is the *Service Agents* Layer where a community of agents have been created to support access to locally available services. Any service can be implemented by reusing the primitives provided at core layer. This provides the system with a high degree of flexibility, allowing user agents access to services through communication with service agents. Services can be added at any moment simply by adding an agent, and all interactions take place through exchanged messages. Therefore, to provide a new service, it is necessary to implement the service agent and a communication protocol. Any user agent that wish to use the new service must exchange messages with the corresponding service agent in accordance with the defined protocol.

Broker agent

The broker agent is a special agent that keeps track of each running service agents and provides any new agent with a list of all locally accessible services. Each agent can access a subset of available services by interacting with service agents on the basis of its credentials. Services can be either a data source or a piece of software code, e.g., an experimental technique. The set of permissions that the agent owns in one place is defined when the agent migrates or is created. If an unqualified agent attempts to use resources, the security component at core layer would intercept upon receiving the agent's message to the service. In such an instance, the message is destroyed and the security component takes precautions against the unqualified agent.

At the present, we are developing three services of general applicability: Web Interface, Wrapper and Ontology Services.

Web Interface

The main task of the Web Interface service is to manage interactions between users and platforms. Through this service, the user can send its own agents, send commands to agents, receive responses and collect any partial or final results. This service agent represents the user's alter ego in the agent system; when the user agent, while performing a task, wants to communicate with the user, it simply sends a message to the Web Interface agent. The Web Interface agent then manages all interactions with the user and responds, if necessary, with a message.

The use of a service agent as a user alter ego is not simply a good choice in principle. In practice, it also implies that the user agent needn't be in possession of all components to manage interactions with the user, and it employs standard communication primitives. These primitives are always provided by the place, so the user agent becomes a very small execution unit implementing features exposed by the core to communicate with other agents, users and services.

Wrapper service

The wrapper is the service that provides access to local resources by abstracting from their nature and providing a uniform representation, for instances using XML. This structural or syntactic abstraction is powered by ontology services which facilitate comprehension of the data semantics. If the ontology service is not available, interpretation of extracted data is left to the agent.

Ontology service

The ontology service manages global and local ontologies based upon a requests from a user agent who must perform a task within a specific application *domain*. The global ontology represents common knowledge shared in all places of the distributed system, while local ontology is knowledge that may be compared to a dialect: knowledge used locally by each site. Integration of two ontologies is used to provide semantics on data that is locally stored and to allow foreign agents to comprehend the meaning of local data. In general, an agent can communicate with the ontology service or it can choose to use only its own ontology. The latter situation may occur when the ontology service is not present. The BioAgent ontology service is being developed upon a similarity algorithm proposed in [4].

2.3 BioAgents Layer

The *BioAgents* layer is populated by user agents, it concerns with the management of user agents. A user agent is an intelligent system capable of flexible autonomous action to satisfy its design *objectives* [6]. To that purpose, user agents should *perceive* their *environment* and respond in a timely fashion to changes that occur within it; agents should be *proactive*; user agents should act not simply in response to their environment but should be capable of exhibiting opportunistic, goal-directed behavior and initiating action where appropriate, user agents should be capable of social interaction, when they deem it appropriate, with other artificial agents and humans in order to complete their own problem solving and help others with their activities.

The behavior of a user agent is defined by its creator or by its owner. There are no constraints on the heterogeneity of running agents, so each user or application could develop and implement its agent and then send it to the place where the BioAgent system is running. Implementation details are given in Section 3. In general, a user agent is created and activated to perform a task on the user's behalf. The Bioscientist's typical experiment, meaningful for a specific domain, is characterized by a sequence of tasks. Any task is defined in terms of activities which must be performed in a featured site, where a set of services (access to data locally stored, the use of experimental techniques) is offered. The site is formally called context. Thus a user agent must know the application *domain* and be aware of the *context* in which specific task will be performed. Execution autonomy is an indispensable feature of a user agent. It can be defined as the ability to decide, based upon its data, which activity to perform next.

2.4 Workflow Layer

The definition of an agent's behavior is a difficult task involving knowledge of programming languages (usually Java, as for our platform) in which a general user is not usually proficient. Therefore, we propose supporting the end user with a user friendly interface suitable for describing the experiment's workflow [12].

In our context, the term workflow means the coordinated execution of multiple tasks or activities. The *Workflow Definition* Layer consists of a set of tools suitable for defining ad-hoc workflow correlated to the coordination of a set of tasks, those used by a bioscientist during an experiment.

A task is defined in terms of activities as follows:

task::= $(\text{task})^+$, $(\text{task})^+$ | activity

activity::= a primitive action | a service

A primitive action is an action directly executable which can be performed on any place. A service is an action supported by services. Any action can be accessed by sending messages to the specific service agent. Obviously, the set of actions provided by services depends on the place. A language called BioAgent-ℓ has been defined to that purpose, and its compiler, which is currently under development, creates and manages user agents. BioAgent-ℓ provides primitives to specify and select the application domain, describe the task and the action's flow control and determine context for each action. Given the high level of language abstraction, workflow definition is possible without knowledge of any programming language. Once the compiling operation has received a few written statements, it will generate user agents suitable for performance of the desired tasks. The user needn't verify the availability of a service or platform; such verifications are encapsulated in the semantic of the language and in the user agent's generation and execution.

In our first prototype, the control flow model used to describe the workflow application is based on the CSP [5] model which is in turn based on a solid mathematical foundation as a formal description technique. Control is exerted by a virtual machine (the engine) that resides on the Workflow layer of the starting place. In the next release, we plan to use Petri net model [7] which, possessing the same expressive power of CSP, provides the advantage of graphical representation. Graphical representation is easier to understand than textual specification. This feature might allow even inexperienced users to create their own nets, given the simplicity of Petri net syntax.

3 BioAgent implementation hints

BioAgent is a platform completely developed in Java based on simplicity, modularity and ease-of-handling [13]. The simplicity is provided by the unique class *Agent* which assigns the basic features to each agent. The class *Agent* is not directly usable, being an abstract class. There are two usable extensions: *UserAgent* and *ServiceAgent*. *UserAgent* is the agent prototype that a user can refer to to perform a task. *ServiceAgent* is the agent prototype that a platform can host to offer a service. For each of the above, agent is defined a proper *SecurityManager*, which denies any user agent access to both local and remote systems' resources (files, and/or, socket, etc.) and which controls the execution of the *ServiceAgent* during access to a system's resources. The agent may clone through the Clone() operation provided by *Agent*, or he may copy it himself. BioAgent allows weak mobility by *go* method.The code mobility operation does not use RMI service, but Java *serialization* and *reflection*. BioAgent uses a dedicated socket port for inter-platforms; if a firewall is present, BioAgent allows tunneling on the port 80. The mobility control has been implemented as planned mobility; that is, the agent itinerary is statically predefined. In particular, the itinerary is mutable and is represented through a tree of locations. The class *Agent* allows communication only between agent-father and agent-son, and is bidirectional – synchronous and asynchronous – based on messages interchange. The methods are *sendToSon*, *sendToFather*, *receiveFromSon* and *receiveFromFather*, and allow communication even when a father or a son has been moved in another platform. Agent() also provides methods for communicating to services. *UserAgent* and *ServiceAgent* can use *sendToService* and *receiveFromService* methods, both synchronous and asynchronous, exclusively addressed to the services locally available.

The BioAgent services are not regularized by the platform. A platform without services allows only "processor service", i.e., computing time. This is because a user agent can employ local resources only through services. Services are uncoupled from the platform, because they are provided as *ServiceAgent*. By using the two methods – *sendToService* and *receiveFromService* – it is possible to provide a client-service feature. To date, the services we have developed use DOM (Document Object Model) objects to describe the content of the message. A query is a DTD instance, that is, an XML document through which the query parameters and query results are transferred. Therefore the creator of a *UserAgent* (presently a bioinformatician; in the future a BioAgent-ℓ compiler) who wishes to use a service must know only its DTD.

4 BioAgent abstract model

A "*software agent* is an *autonomous* process capable of reacting to, and initiating changes in, its *environment*, possibly in collaboration with users and others agents"[6].

To highlights the features that a bioagent must possess to carry out his task, we must define the abstract model which describes the different user agent roles. We can hypothesize that a general bioscientist prefers to work using the highest level of global abstraction (Ontological level), while others, the bioinformatics, less proficient in information and communication technology, choose to develop agents directly using, for example, Java programming language. Therefore, the model we refer to consists of two abstraction levels:

- *the Ontological level - Domain*, which models the applicative potential expressed in terms of global knowledge, and the common conceptualization of an application's domain, e.g., the *Global Ontology*. If the conceptualization of the application's domain is only locally known, we have called it *Local Ontology*.
- *the environmental level - Context*: It models the environment in which the agent is going to work[15]; Common data schema and services are directly correlated to the physical environment (data, access methods and communication primitives available at any local place).

A workflow is defined at the ontological level while bioagents are created at the environmental level. The conceptual model used to describe an application domain is completely independent of the model used by an agent to perform in its environment. In our framework, the application domain is defined in terms of Global Ontology [4]. An ontology is a formal specification of a shared conceptualisation, that is, the knowledge structure that describes the semantics of an information source by using a lexicon. A lexicon consists of a finite set of semantically meaningful concepts, and a finite set of the relationships feasible between concepts. At the environmental level, the context describes how the application domain has been concretely represented in terms of syntactic aspects of data structures and access methods, depending on the features of the place. The bioagent must be able to use both the *global ontology* which describes the application domain in terms of a common lexicon, and the *local ontology* possibly defined in the place to describe local concepts by a specific lexicon. User agents usually work with support from the *ontology service*.

4.1 An application

Usually, a biological data analysis experiment is defined for a specific domain and consists of a set of concurrent activities.

For example, the goal of the microarray user is to obtain meaningful answers to his/her starting biological hypothesis. This goal can be accomplished by suitable cluster analysis of a set of distinct gene-expression experiments (a series). The clustering procedure can be applied to selected series to group similar genes and/or similar experiments. The distributed nature of gene-expression data repositories poses a fundamental need to the single experimenter: gaining access to other labs' experimental data sets, to use them in a multi-site cluster analysis. Each experiment consists of a set of tasks and activities executed by following a specific method; each task manipulates distributed data by using ad hoc algorithms and analysis techniques. Some tasks may be processed in sequence others in parallel, and rarely they are independent from the others. The whole analysis process is represented by the experiment's workflow.

Let us suppose that a bioscientist is looking for:

- the genetic profile of a set of genes (8, 10 thousand genes) which have been subjected to specific experimental conditions such as, for example, a variation in the genetic profile following treatment with specific pharmaceutical substances, or variations of the genetic profile following induction of "apoptosi" of other molecular phenomena; or variations of the genetic profile between tumor tissue and comparable normal control tissue where access to the research is obtained through key words like substance X, "apoptosi", breast cancer, etc.).

- In such instances, the researcher must be able to view all relevant experiments involving microarrays and breast cancer, for example; or view locally what may have been done globally for a given experiment with that particular type of spotting or hybridization and so on.

  1) spotting (est sequence, orf sequence, small oligos); 2) Type of spotting (human, mouse, yeast, etc.); 3) labeling (cy3; cy5; ect.); 4) Type of labeling (DNA or RNA); 5) type of hybridization (manual, automatic); 6) normalization (internal spike, housekeeping gene, etc); 7) data analysis (average, median, moda confidence, etc.)

We plan to define the experiment in terms of the following tasks and activities:

task1: search for information about concepts (breast cancer, apoptosis), by using ontology [3, 16] in the microarray domain, *then* select all interest data repositories)

task2: go to each one of the selected sites, extract experiments, *then* cluster all experiments by "gene" or "by experiment"

task3: statistical analysis and visualization of the results

All defined tasks will originate a bioagent. A bioagent is created to work in a specific *domain*, his starting *knowledge* (global ontology). He must compute a task by performing a set of actions in a specific *environment* (local ontology, services and data). In the above specific experiment, for example the bioagent created to perform task1 will work on the "microarray" domain; he will own a microarray global ontology (if such exists); he will start by travelling the network to select sites of interest by using the ontology service and the wrapper service [4, 18]. Preliminary results on this example can be found in [18].

5 Conclusions

BioAgent provides an easy to use mobile agent platform specifically created for a biological application domain. The layered architecture of the system affords definition of a BioAgent model based on different levels of abstraction according to the workflow used to describe the experiment of a researcher, and to the user agents created to support the single task characterizing the steps of experiments. BioAgent has been designed based upon a concept of modularization of service agents which allows agents to easily plug-in to new services without involving the end user. Experiments on real data have demonstrated how off-line search and decentralized analysis can reduce network traffic. BioAgent proposes solutions to many problems of the bioinformatics community by providing opportunities to search off-line, to share data efficiently, to collect information, integrate and manage scientific results from multiple sources and transform the actual fragmented scenario into a solid, homogeneous, and user-friendly environment.
Our next step, is to complete the development of the two architectural highest layers functionalities and to prove the system in a complex real applications. Last but not least, we aim to compare efficiency and effectiveness of our mobile system with those not mobile but proposed to support automated genomic annotation problem [9, 8].

acknowledges

The authors would like to acknowledge Luca Toldo and Mauro Agneletti for useful discussions, as well as the students – especially Diego Bonura, Francesca Piersigilli and Davide Lorenzoli for their invaluable contributions during the system's development stage.

References

[1] B. Brewington, R. Gray, K. Moizumi, D. Kotz, G. Cybenko, and D. Rus, *Mobile agents for distributed information retrieval*, in Intelligent Information Agents, M. Klusch, ed., Springer-Verlag, 1999.

[2] P. Ciancarini, A. Giovannini, and D. Rossi, *Mobility and Coordination for*

Distributed Java Applications, in Recent Advances in Distributed Systems, S. Krakowiak and S. Shrivastava, eds., vol. 1752 of Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 402–425.

[3] G. O. Consortium, *Gene ontology: tool for the unification of biology*, Nature Genetics, 25 (2000), pp. 25–29.

[4] R. Culmone, G. Rossi, and E. Merelli, *An ontology similarity algorithm for bioagent*, in NETTAB Workshop on Agents nd Bioinformtics, Bologna, 2002.

[5] C. Hoare, *Communicating sequential processes*, in Communications of the ACM, August 1978, pp. 666–677.

[6] N. Jennings and M. Wooldridge, *Application of intelligent agents*, in Agent Technology: Foundations, Applications, and Markets, Springer-Verlag, 1998.

[7] K. Jensen, *Coloured Petri Nets*, Springer-Verlag, 1992.

[8] M. K. Bryson, M. Luck and D. Jones, *Applying agents to bioinformatics in geneweaver*, in Cooperative Information Agent IV, vol. 1860, Lecture Notes in Artificial Intelligence, 2000, pp. 60–71.

[9] X. Z. K. Decker and C. Schmidt, *A multiagent system for automated geneomic annotation*, in Proccedings of the 5th Intl. Conference on Autonomous Agents, 2001.

[10] D. Lange and M. Oshima, *Programming and Deploying Java Mobile Agent with Aglets*, Addison-Wesley, 1998.

[11] Mednick and Donovan, *Operating Systems*, Mc Graw Hill, 1978.

[12] M. Merz, B. Lieberman, and W. Lamersdorf, *Using mobile agent to support interorganizational workflow management*, Applied Artificial Intelligence, 11 (1997), pp. 551–572.

[13] U. of Camerino, *Bioagent project page. http://www.bioagent.org.*

[14] U. of Stuttgart, *Mole project pages, http://www.mole.informatik.uni-stuttgart.de.*

[15] A. Omicini, *Towards a notion of agent coordination context*, in Process Coordination and Ubiquitous Computing, C. Lee, ed., CRC Press, 2002, pp. 187–200.

[16] S. B. N. P. R. S. P. Baker, C. Goble and A. Brass, *An ontology for bioinformatics applications*, Bioinformtics, 1999 (15), pp. 510–520.

[17] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Descipline*, Prentice-Hall, 1996.

[18] L. Soverchia, M. Angeletti, R. Culmone, and E. Bartocci, *Dna-microarray data integration using the bioagent architecture: a new approach for researchers cooperation in the genome era*, in NETTAB Workshop on Agents nd Bioinformtics, Bologna, 2002.

[19] N. Suri, *An overview of the nomads mobile agent system*, in Proceedings of ECOOP 2000, 2000.