Proceedings of the International MultiConference of Engineers and Computer Scientists 2008 Vol I
IMECS 2008, 19-21 March, 2008, Hong Kong

Developing a Multiagent System for Integrating Biological Data Using JADE

Faheema Maghrabi, Hossam M. Faheem, and Tayseer Hassan**,** and Zaki T. Fayed

Abstract—**Biological data has been rapidly increasing in voluminous data of different data sources. To query multiple data sources manually on the internet is a time consuming task for the biologists. Therefore, systems and tools that facilitate searching multiple biological data sources are needed. Traditional approaches to build distributed or federated systems do not scale well to the large, diverse, and growing number of biological data sources. Recent Internet systems such as World Wide Web browsers allow users to search through large numbers of data sources, but provide very limited capabilities for locating, combining, processing, and organizing information. A promising approach to this problem is to provide access to the large number of biological data sources through a multiagent technology, where a set of agents can cooperate with each other to retrieve relevant information from different biological web databases. The proposed system uses a mediator based integration approach with domain ontology which used as a global schema. The proposed system is developed using JADE (Java Agent DEvelopment Framework) which is a software development framework aimed at developing multi-agent systems and applications conforming to FIPA (Foundation for of Intelligent Physical Agent) standards for intelligent. In this paper we develop a multiagent system that responds to different user queries to multiple heterogeneous biological databases. The system works as a middleware between users and different biological databases.**

Index Terms— **Biological data integration, Biological ontology, Multiagent technology, JADE.**

I. Introduction

Recent advances in laboratory technology have resulted in massive amounts of biological data that are often deposited in web databases. Clearly, access to this data is very important to biological researchers. However, heterogeneity among biological databases due to the incompatibilities in data formats, data representations, and data source schema has impeded the accessibility to these databases. Each of the biological databases has its own interfaces and control languages, and represents information using conflicting data models and formats. The goal of the proposed system is to hide the large variety of local or remote biological databases and the disparity of their interfaces. A user should see only one interface and be able to query the system by specifying what it wants to know without a detailed knowledge where relevant information is located, what its representation is like, and how the biological databases interfaces must be handled.

Three fundamental approaches have been used to address the challenges associated with the incompatibilities among biological databases: data warehousing integration, information linkage integration, and mediator-based integration. Data warehousing consists of materializing the data from multiple data sources into a local warehouse and executing all queries on the data contained in the warehouse rather than in the actual sources. Data warehousing suffers from a lack of scalability when considering the exponential growth of biological databases. The Information linkage integration is motivated by the fact that many of data sources on the web are browsed instead of queries. The integration happens through links and applies to any collection of data sources which can be seen as a set of pages with their interconnections and specific entry point. Information linkage takes advantage of distributed resources. However, maintaining and updating the static links between various databases is a challenge. Furthermore, the only queries that can be answered by information linkage based systems are those that are within the scope of the pre-existing static links [1, 2].

The proposed system is based on the third approach, mediator-based integration, which establishes a transparent access to heterogeneous data sources without physically copying them into a single data repository. This class of integration systems can be divided into two subclasses: Local As View (LAV) and Global As View (GAV). In LAV, there is no global schema, and the user needs to specify the component databases in the query by using a multidatabase query language. One of the disadvantages of LAV is that the component databases are not transparent to the user. This approach is used by BioKleisli [3]. In GAV, a global data schema is constructed, and queries are expressed in this unified schema. The global schema integrates all the component schemas therefore, the component databases are transparent to the user. TAMBIS [4] and BACIIS [5] are examples of GAV, where they use ontology as a conceptual model to integrate multiple biological web databases.

Domain ontology and mapping schema are the two main components of the proposed system. The **domain ontology** can be used to define a common controlled vocabulary and to semantically define databases. It is designed with hierarchal structure describing the biological concepts and the relationship between them. When integrating

F. Maghrabi graduated in Faculty of Computer & Information Sciences Ain Shams University, Computer science department (e-mail: bosycs@ yahoo.com).
H.M.Faheem, , Associate Professor, Computer Systems department, Faculty of Computer and Information Sciences, Ain Shams University, Director of EUN (e-mail: hmfaheem@ieee.org).
T.Hassan, Teacher in Faculty of Computer & Information Sciences ,Ain Shams University, Information system department (email:taysir_soliman@hotmail.com)
Z.T..Fayed, Associate Professor, Computer science department, Faculty of Computer & Information Sciences, Ain Shams University.

heterogeneous databases, some issues have to be addressed: semantic and syntactic variability. Resolving semantic variability consists of adequately interpreting and cross relating information stored in different databases. Syntactic variability arises from the heterogeneity of the database schema, data models, and query processing approaches. Domain ontology has been used successfully as domain model in several general-purpose information integration systems. The domain ontology in this architecture serves as a global schema and plays a major role in resolving both semantic and syntactic variability. Domain ontology can be used for guiding user through query formulation. It is also used in data source selection utilizing mapping schema, and in rewriting the user query into smaller subqueries.

BACIIS Ontology **(BAO)** is an example of biological ontology. The ontology developed in BAO has three dimensions: object, property, and relation. In the ontology of BAO, the properties of an object are defined as a property class which occupies a position in the property hierarchy tree. A relation refers to the association between the two concepts. In BAO, classes under the object and property dimensions are arranged into a hierarchy tree based on the relation "is-a-subclass-of." For example, GENE is a subclass of the class NUCLEIC-ACID in the object hierarchy. Similarly, "base-count" is a subclass of "NUCLEIC-ACID-SEQ-INFO" in the property hierarchy. The relation "has property" is used for the object classes and property classes as in the case of the class NUCLEIC-ACID in the object hierarchy and the class "NUCLEIC-ACID-INFO" in the property hierarchy. The relation "is-a-subsetof" is used for property classes or object classes with parent-descendant relationships. The relations "regulate" and "source-of" are used for object classes that are neither parents nor descendants of each other [5].

The **mapping schema** is used to describe each database participating in the integration. In each mapping schema a mapping between data schema of web database and domain ontology is made. This mapping includes concepts and terms from domain ontology that are relevant to specific database. It contains metadata of how to query and extract data from the web interface of specific web database. Each mapping schema consists of the metadata which includes general information about the web database such as database name, the input data types accepted, and the output data types generated by the corresponding web database. These data types are expressed by using ontology terms. The mapping schema also used to select the component database that can respond to the given subquery.

The proposed system is based on agent technology. The following requirements are best met within an agent-based environment; modularity, extensibility, flexibility, and declarative forms of communications. **Modularity** allows new components to be added or removed without affecting the operation of other system parts. **Extensibility** allows for new elements to be easily added to the system. **Flexibility** provides the ability to deal with the dynamic state of the system, e.g. availability of data sources. **Declarative forms of communication** allow the communication between components into the system to deal primarily with information and knowledge. Thus, communication protocols based on known agent communication languages are more appropriate for our purpose [6].

This paper is organized as follows: section II introduces the overview of agent technology**,** section III discusses why agent technology is a promising approach to address the biological data integration problem, section IV discuss JADE framework, section V introduce components of proposed system, section VI presents example and results, and section VII augments some concluding remarks.

II. OVERVIEW OF AGENT TECHNOLOGY

Agent may exhibit three important general characteristics: *autonomy*, *adaptation,* and *cooperation.* By "autonomy" we mean that agents have their own agenda of goals and exhibit goal-directed behaviour. They are not simply reactive, but can be proactive and take initiatives as they deem appropriate. Adaptation implies that agents are capable of adapting to the environment, which includes other agents and human users, and can learn from the experience in order to improve themselves in a changing environment. Cooperation and coordination between agents is probably the most important feature of multi-agent systems. Unlike those stand-alone agents, agents in a multi-agent system collaborate with each other to achieve common goals. In other words, these agents *share* information, knowledge, and tasks among themselves [7].

Agent programs have four major types they are: simple reflex agents, agents that keep track of the world, goal-based agents, and utility-based agents. A simple reflex agent works by finding a rule whose condition matches the current situation (as defined by the percept) and then doing the action associated with that rule. An agent that keeps track of the world works by finding a rule whose condition matches the current situation (as defined by the percept and the stored internal state) and then doing the action associated with that rule. A goal-based agent selects any action (from a set of actions) that can achieve the goal. Utility based agent selects the best action (from a set of actions that can achieve the goal) to achieve the goal in order to maximize the happiness [8].

III. AGENT TECNOLOGY AND BIOLOGICAL DATA INTEGRATION

Three important aspects of biological data integration are distribution, autonomy and heterogeneity. **Distributions**, in most cases data sources are distributed. The user need not know the location and other details of each available data resource. **Autonomy**, it is very often the case that integrated resources belong to different organizations or research groups. While most people are willing to share their data, they do not want to lose control over decisions for their data source. Thus, the developers of an integrated system do not usually have any control over the underlying systems, which are autonomous. **Heterogeneity**, in an open and diverse environment it is very common that some or all of the data sources are different from each other. Integrating heterogeneous databases involves extra work so as to ensure the correct relationship of data between the information systems. Agents naturally cover the fundamental aspects of data integration. [9].

IV. JADE

JADE is free software and is distributed by Telecom Italia Lab (TILAB), as open source software under the terms of the LGPL (Lesser General Public License Version 2). It is implemented in Java and aims to make easier the development of multiagent systems by providing a comprehensive set of services and an API to support the development of agents in compliance with the FIPA specifications. The JADE framework provides an API with several classes that represent common entities in multiagent system, supporting the definition of agents, agents behaviours, interaction protocols, and so on. In the time of vocabulary and semantics definition of the agents' communication content, a solution based on Java objects usage as the content of the messages exchanged among the agents has been adopted. The JADE utilization has been essential to the agents implementation, simplifying and significantly speeding up the process. JADE includes a **runtime environment** where JADE agents can "live" and that must be active on a given host before one or more agents can be executed on that host. JADE also includes a suite of **graphical tools** that allow administrating and monitoring the activity of running agents. JADE has several properties such as has good documentation, very good GUI, acceptance of users, used in many development projects, standard FIPA, very good security features, and various communication protocols[10].

A. Containers and platforms

Each running instance of the JADE runtime environment is called a ***Container*** as it can contain several agents. The set of active containers is called a ***Platform***. A single special ***Main container*** must always be active in a platform and all other containers register with it as soon as they start. It follows that the first container to start in a platform must be a main container while all other containers must be "normal" (i.e. non-main) containers and must "be told" where to find (host and port) their main container (i.e. the main container to register with). You don't have to know how the JADE runtime environment works, but just need to start it before executing your agents. [10]

B. AMS and DF

Besides the ability of accepting registrations from other containers, a main container differs from normal containers as it holds two special agents (automatically started when the main container is launched). The first is **AMS** (Agent Management System) that provides the naming service (i.e. ensures that each agent in the platform has a unique name) and represents the authority in the platform (for instance it is possible to create/kill agents on remote containers by requesting that to the AMS The second agent is **DF** (Directory Facilitator) that provides a Yellow Pages service by means of which an agent can find other agents providing the services he requires in order to achieve his goals.

C. Creating and terminating agent

Creating a JADE agent is as simple as defining a class extending the jade.core.Agent class and implementing the setup () method .The setup () method is intended to include agent initializations. The actual job an agent has to do is typically carried out within "behaviors". Each agent is identified by an "agent identifier" represented as an instance of the jade.core.AID class. The getAID() method of the Agent class allows retrieving the agent identifier. An AID object includes a globally unique name plus a number of addresses. The name in JADE has the form *<nickname>@<platform-name>* so that an agent called UIA living on a platform called *P1* will have *UIA@P1* as globally unique name. In order to make it terminate its doDelete() method must be called. Similarly to the setup() method that is invoked by the JADE runtime as soon as an agent starts and is intended to include agent initializations, the takeDown() method is invoked just before an agent terminates and is intended to include agent clean-up operations.

D. Agent behaviours

The actual job an agent has to do is typically carried out within "behaviours". A behaviour represents a task that an agent can carry out and is implemented as an object of a class that extends jade.core.behaviours.. In order to make an agent execute the task implemented by a behaviour object it is sufficient to add the behaviour to the agent by means of the addBehaviour() method of the Agent class. Behaviours can be added at any time: when an agent starts (in the setup() method) or from within other behaviours. Each class extending Behaviour must implement the action() method, that actually defines the operations to be performed when the behaviour is in execution and the done() method (returns a boolean value), that specifies whether or not a behaviour has completed and have to be removed from the pool of behaviours an agent is carrying out.

E. Agent communication

One of the most important features that JADE agents provide is the ability to communicate. Each agent has a sort of mailbox (the agent message queue) where the JADE runtime posts messages sent by other agents. Whenever a message is posted in the message queue the receiving agent is notified. If and when the agent actually picks up the message from the message queue to process it is completely up to the programmer however.

Messages exchanged by JADE agents have a format specified by the Agent Communication Language (ACL) defined by the FIPA.international standard for agent interoperability. This format comprises a number of fields and in particular:

• The ***sender*** of the message.
• The list of ***receivers***
• The communicative intention (also called "***performative***") indicating what the sender intends to achieve by sending the message. The performative can be REQUEST, if the sender wants the receiver to perform an action, INFORM, if the sender wants the receiver to be aware a fact, QUERY_IF, if the sender wants to know whether or not a given condition holds, CFP (call for proposal), PROPOSE, ACCEPT_PROPOSAL, REJECT_PROPOSAL, if the sender and receiver are engaged in a negotiation, and more.
• The ***content*** i.e. the actual information included in the message (i.e. the action to be performed in a REQUEST message, the fact that the sender wants to disclose in an INFORM message …).

• The content ***language*** i.e. the syntax used to express the content (both the sender and the receiver must be able to encode/parse expressions compliant to this syntax for the communication to be effective).
• The ***ontology*** i.e. the vocabulary of the symbols used in the content and their meaning (both the sender and the receiver must ascribe the same meaning to symbols for the communication to be effective).
• Some fields used to control several concurrent conversations and to specify timeouts for receiving a reply such as ***conversation-id***, ***reply-with***, ***in-reply-to***, ***reply-by***.

F. GRAPHICAL TOOLS

To support the difficult task of debugging multi-agent applications, some tools have been developed. Each tool is packaged as an agent itself, obeying the same rules, the same communication capabilities, and the same life cycle of a generic application agent [11]. There are four tools we discuss in the following sections.

- *Remote Monitoring Agent: The* Remote Monitoring Agent (RMA) allows controlling the life cycle of the agent platform and of all the registered agents. The distributed architecture of JADE allows also remote controlling, where the GUI is used to control the execution of agents and their life cycle from a remote host.
- *Dummy Agent: The* DummyAgent tool allows users to interact with JADE agents in a custom way. The GUI allows composing and sending ACL messages and maintains a list of all ACL messages sent and received. This list can be examined by the user and each message can be viewed in detail or even edited. Furthermore, the message list can be saved to disk and retrieved later. Many instances of the DummyAgent can be started as and where required.
- *DF GUI:* A GUI of the DF can be launched from the Tools menu of the RMA. This action is actually implemented by sending an ACL message to the DF asking it to show its GUI. Therefore, the GUI can just be shown on the host where the platform (main-container) was executed. By using this GUI, the user can interact with the DF: view the descriptions of the registered agents, register and deregister agents, modify the description of registered agent, and also search for agent descriptions.
- *Sniffer Agent: As* the name itself points out, the Sniffer Agent is basically a FIPA-compliant Agent with sniffing features. When the user decides to sniff an agent or a group of agents, every message directed to/from that agent / agent group is tracked and displayed in the Sniffer Agent's Graphical User Interface (GUI). The user can view every message and save it to disk.

V. COMPONENTS OF THE PROPOSED SYSTEM

The proposed system consists of biological ontology, mapping schema, and multiagent system.

A. Biological ontology

Due to the cost of designing the ontology we deployed BACIIS ontology as biological domain ontology. BACIIS ontology is less complex than the concept of TAMBIS ontology. Moreover, BACIIS concepts are easier to understand and could be modified consequently. BACIIS ontology developed using protégé-2000(Protégé-2000 is an extensible, platform independent ontology and knowledge-base editor, developed by Stanford Medical Informatics at the Stanford University School of Medicine. Protégé is available as free software) and then converted to JADE compatible java classes using *jadejessprotege* (plugin for protégé). Fig.1 show partial structure of BAO designed using protégé.

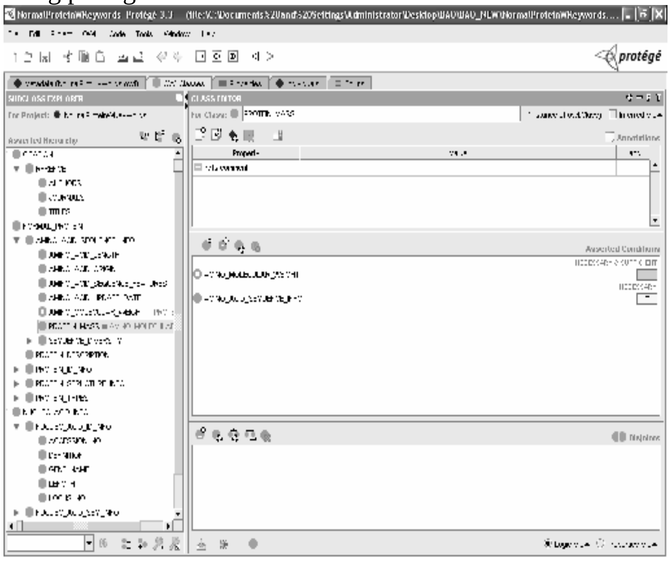


Fig.1 Partial of BACIIS ontology designed using protégé

B. Mapping schema

We initially used two biological databases SWISSPROT (protein) and GENBANK (Gene), we mapped each term in database to BACIIS ontology concepts. By mapping operation we solve heterogeneity between databases. for example, to refer to the organism name, the database GENBANK uses "ORGANISM" and the SWISSPROT use "OS". The ontology is used to map all the different terms used by the remote databases to a single term (see Fig.2). Each database has its own mapping schema .we can easily add other biological databases only by creating mapping schema and wrapper agent for this database.

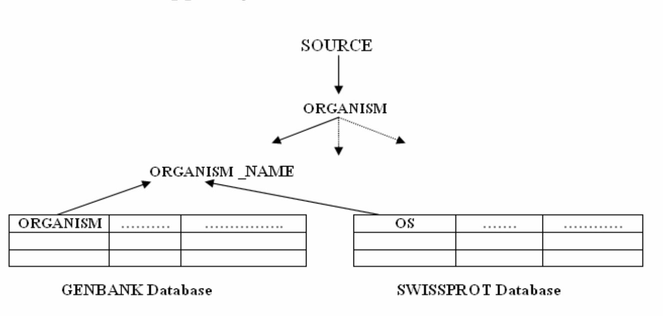


Fig.2 Mapping operation

C. Multiagent system

As shown in Fig.3 the proposed framework includes the following agents:

- The User Interface Agent (UIA) guides user to construct ontology based query, and then sends this query to the Query Planner Agent, and receives result data from Execution Agent.
- The Query Planner Agent (QPA) decomposes this query into a list of subqueries, and then sends these subqueries to the Mapping Agent, then receives a list of appropriate

databases from mapping agent, then constructs the execution path for this query (select the best execution path), and then sends this execution path to the Execution Agent.

- The Mapping Agent (MA) prepares a list of appropriate databases using mapping schema, and then sends this list of databases to the Query Planner Agent.
- The Execution Agent (EA) invokes the Wrapper Agent associated with each web database that corresponds to a given subquery, and collects result data from wrapper Agents then sends to User Interface Agent which displays result.
- The wrapper Agents (WA) retrieves the result data from remote databases using mapping schema of this database. The mapping schema contains rules of how to extract data from web interface of this database, and then send these results data to the Execution Agent. Table I shows a brief description for each agent, percepts, actions, goals, and type.

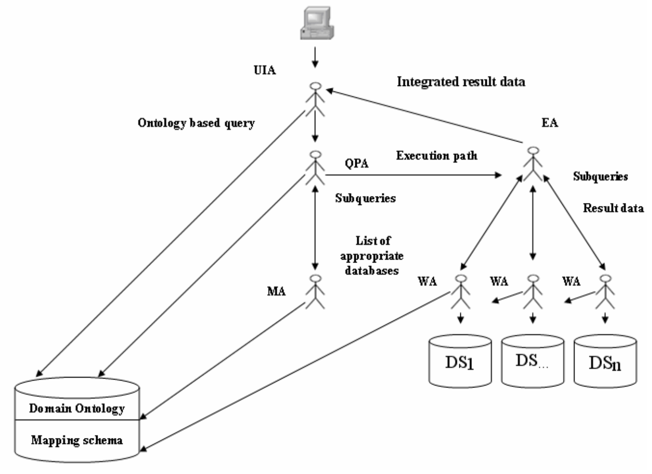


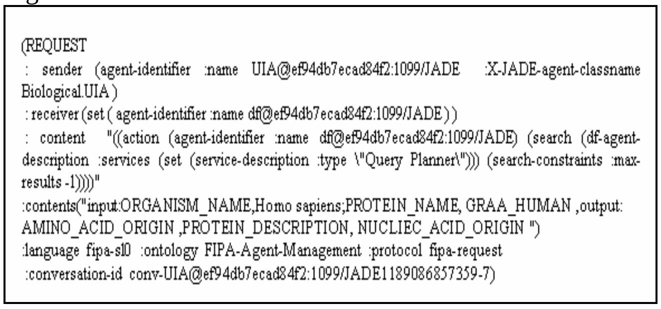
Fig.3 Multiagent framework

VI. EXAMPLE

Consider the following user query: Find the sequence and description of "GRAA_HUMAN" protein of "Homo sapiens" organism, also find sequences and length of their coding gene.

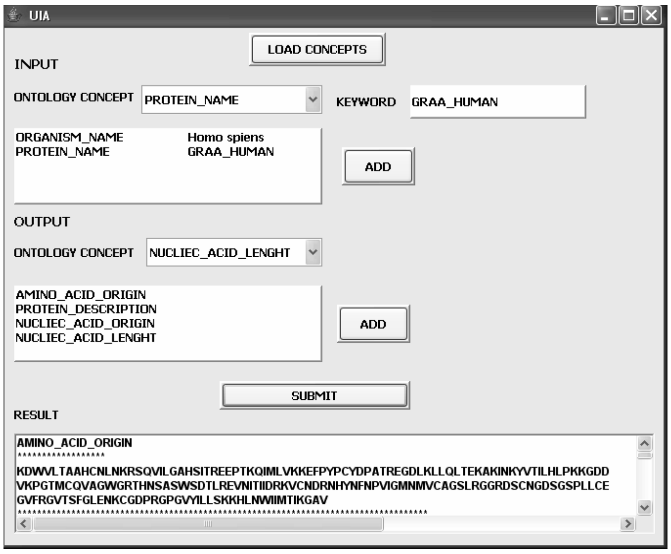
UIA guides user in constructing the query which consist of two parts; input and output. In input part, user selects input type (BACIIS ontology concept) and types keyword. In output part, user selects output type (BACIIS ontology concept). According to the given example , the input query are " GRAA_HUMAN" of input type are PROTEIN_NAME and "Homo sapiens" of input type ORGANISM-NAME .the output query are AMINO_ACID_ORIGIN, PROTEIN_DESCRIPTION , and NUCLIEC_ACID_ORIGIN(see Fig.4).

The UIA then sends **REQUEST** message to QPA such as in Fig 5.



```
(REQUEST
 :  sender  (agent-identifier  :name  UIA@ef94db7ecad84f2:1099/JADE    :X-JADE-agent-classname
Biological.UIA )
 : receiver (set ( agent-identifier :name df@ef94db7ecad84f2:1099/JADE ) )
 :  content   "((action  (agent-identifier  :name  df@ef94db7ecad84f2:1099/JADE)  (search  (df-agent-
description  :services  (set  (service-description  :type  \"Query  Planner\"))) (search-constraints  :max-
results -1))))"
:contents("input:ORGANISM_NAME,Homo sapiens;PROTEIN_NAME, GRAA_HUMAN ,output:
AMINO_ACID_ORIGIN ,PROTEIN_DESCRIPTION, NUCLIEC_ACID_ORIGIN ")
:language  fipa-sl0  :ontology  FIPA-Agent-Management  :protocol  fipa-request
 :conversation-id  conv-UIA@ef94db7ecad84f2:1099/JADE1189086857359-7)
```

Fig.5 Request message to QPA from UIA



INPUT
ONTOLOGY CONCEPT PROTEIN_NAME KEYWORD GRAA_HUMAN

ORGANISM_NAME	Homo spiens
PROTEIN_NAME	GRAA_HUMAN

OUTPUT
ONTOLOGY CONCEPT NUCLIEC_ACID_LENGHT

AMINO_ACID_ORIGIN
PROTEIN_DESCRIPTION
NUCLIEC_ACID_ORIGIN
NUCLIEC_ACID_LENGHT

RESULT

```
AMINO_ACID_ORIGIN
*****************
KDWVLTAAHCNLNKRSQVILGAHSITREEPTKQIMLVKKEFPYPCYDPATREGDLKLLQLTEKAKINKYVTILHLPKKGDD
VKPGTMCQVAGWGRTHNSASWSDTLREVNITIIDRKVCNDRNHYNFNPVIGMNMVCAGSLRGGRDSCNGDSGSPLLCE
GVFRGVTSFGLENKCGDPRGPGVYILLSKKHLNWIIMTIKGAV
*********************************************************************************
```

Fig.4 GUI of UIA

QPA receive **REQUEST** message that contain the ontology concepts from UIA and then decompose it using mapping schema and biological ontology. QPA then sends a **REQUEST** message contains these subqueries to MA which maps each subquery to biological database, and then sends **INFORM** message includes a list of appropriate databases to QPA which constructs the execution path (see Fig 6).

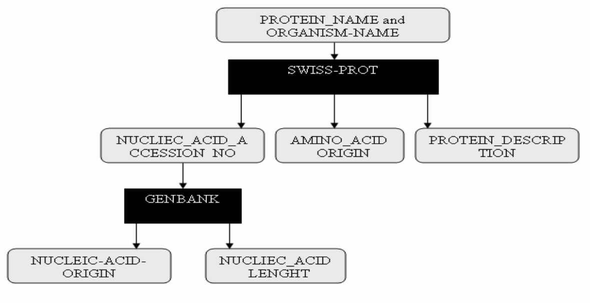


Fig.6 Execution path

QPA sends INFORM message to EA contains the execution path, The EA sends REQUEST message contains the subquery for each WA associated with databases according to dependencies in execution path. The WA use the mapping schema to map ontology concept to corresponding term in database, and then submit the subquery after transform it to the form compatible to the web interface of specific database(using the mapping schema). And then sends INFORM message contain result to EA which collects and integrate all result received from WAs, EA then sends INFORM message contain integrated result set for UIA which display it to user. Fig.7 shows sniffer agent GUI which display message passing between agents .In this figure a messages directed to/from DF agent. DF agent receive REQUEST message for information, such as agent name, about any agent registered and send INFORM message contain this information.

VII. CONCLUSION

Agent-oriented techniques are being increasingly used in several applications. They are ideally qualified for developing complicated and distributed software systems. In fact, agents enhance software modularity, maintainabilit

Table. I Agent description

Agent name	Percepts	Actions	Goals	Type	Explanation
UIA	Input query sent by the user, the integrated result set sent by the **RA**.	Guide user to construct ontology based query and output the integrated result set to user.	Construct ontology based query, output the integrated result set to user.	Simple reflex agent.	This agent selects actions on basis of current percept (input query) only.
QPA	Ontology based query sent by the **UIA**, list of appropriate databases sent by **MA**.	Decompos e input query into smaller subqueries .	Construct the execution path.	Utility based agent.	This agent select the best action (best execution path) that achieves the goal.
MA	Subqueries sent by the **QPA**.	Map each subquery to a specific data source using mapping schema	Find list of appropriat e data sources that submit subqueries .	Agent that keeps track of world.	This agent select action based on an internal state (mapping schema) and current percept (subquery).
EA	Execution plan sent by the **QPA**. Result data sent by the **WAs**.	Invoke the **WAs** associated with each web database. Collects query results data from **Was** and send it to **UIA**.	Invoking the **WAs** associated with each web database according to dependenc ies in execution path. Collects query results data from **WAs** and send it to **UIA**.	Simple reflex agent.	This agent selects actions on basis of current percept (execution path, result data) only.
WA	Ontology based Subqueries sent by the **EA**.	Retrieves the result data from remote databases.	Submit the subquery and retrieves the result data.	Agent that keeps track of world.	This agent select action based on an internal state (mapping schema) and current percept (subquery).

and reusability . In this paper, we proposed a multiagent system that responds to different biological queries according to its biological domain ontology. A set of agents can cooperate with each other to retrieve relevant information from different biological web databases. The proposed system resolves the incompatibilities in data formats, data representations, and mapping schema of biological web databases. It also reduces the overhead associated with changing the existing or adding different data sources. Mediator based integration approach with domain ontology has been deployed. One of the main advantages of this approach is that it does not require component databases to be physically combined into one database. The domain ontology serves as a global schema and plays a major role in resolving both semantic and syntactic variability of biological web databases. The proposed system is developed using JADE platform.

We believe that the proposed system is a step towards a complete multiagent-based system for bioinformatics applications.

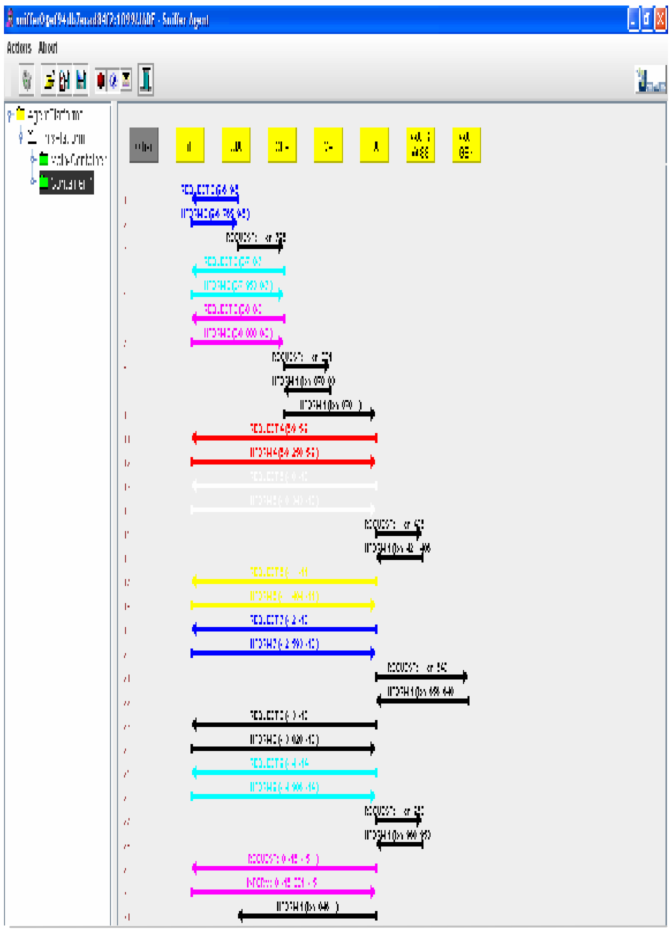


Fig.7 Sniffer agent GUI

REFERENCES

[1] Z.B Miled, Y.W.Webster, and Y.Liu, " An Ontology for Semantic Integration of Life Science Web Databases", International Journal of Cooperative Information Systems, World Scientific Publishing Company, Vol.12 No.2, 2003, pp.275-294.

[2] T.Hernandez, and S.Kambhampati, " Integration of biological sources: current systems and challenges ahead", ACM SIGMOD Record , ACM Press ,Vol.33 No 3, 2004, pp.51 – 60 .

[3] S.B.Davidson, C.Overton, V.Tannen, and L.Wong, “BioKleisli: a. digital library for biomedical researchers", International Journal on Digital Libraries, Springer Berlin, Vol.1 No.1, 1997, pp.36-53.

[4] P.G.Baker, A.Brass, S.Bechhoferb, C.Goble, N.Paton, and R.Stevens., " TAMBIS - Transparent Access to Multiple Bioinformatics Information Sources", Proceedings of the 6th International Conference on Intelligent Systems for Molecular Biology, AAAI Press, 1998, pp.25-34.

[5] Z.B.Miled, N.Li, and O.Bukhres, " BACIIS: Biological and Chemical Information Integration System", Journal of Database Management, Idea Group Inc, Vol.16 No.3, 2005, pp. 72-85.

[6] E.Burger, J.Link,, and O.Ritter, "A Multi-Agent Architecture for the Integration of Genomic Information", First International Workshop on Intelligent Information Integration (in conjunction with KI97), 1997

[7] Y.Peng, T.Finin, Y.Labrou, B.Chu, J. Long, W. J. Tolone, and A. Boughannam, "A Multi-Agent System for Enterprise Integration, International", Journal of Agile Manufacturing, UMBC eBiquity, Vol.1 No 2, 1998, pp.213-229.

[8] S.Russell, and P.Norvig, “Artificial Intelligence: A Modern Approach”, Printice Hall Inc, New Jersey, USA, 1995.

[9] K.A.Karasavvas, R.Baldock, and A.Burgera, "Bioinformatics integration and agent technology", Journal of Biomedical Informatics, Elsevier Science, Vol 37 No.3, 2004, pp.205 - 219 .

[10] G. Caire," JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS", available at http://jade.tilab.com.

[11] F.Bellifemine, G.Caire, T.Trucco, G.Rimassa,R.Mungenast," JADE Administrator’s GUIDE ", available at http://jade.tilab.com .