Deploying Mobile Agents in Distributed Data Mining

Xining Li and JingBo Ni

State Key Laboratory of Novel Software Technology (Nanjing University)
Department of Computing and Information Science, University of Guelph, Canada
xli@cis.uoguelph.ca, jni@uoguelph.ca

Abstract. Mining information from distributed data sources over the Internet is a growing research area. The introduction of mobile agent paradigm opens a new door for distributed data mining and knowledge discovery applications. In this paper, we present the design of a mobile agent system, which couples a logic language based application programming interface for service discovery and database access. Our proposal aims at implementing system tools to enable intelligent mobile agents to search for distributed data services, to roams the Internet for accessing distributed data sites, to discover patterns and extract useful information from facts recorded in databases, to generate global data model through the communication and aggregation of local results, and to overcome the barriers posed by network congestion, poor security and unreliability.

Keywords: Mobile agent, distributed data mining, service discovery, database service.

1 Introduction

Mobile agent systems bring forward the creative idea of moving user defined computations – agents, towards network resources, and provide a whole new architecture for designing distributed systems. Distributed data mining (DDM) is one of the important application areas of deploying intelligent mobile agent paradigm [1], [2]. Most existing DDM projects focus on approaches to apply various machine leaning algorithms to compute descriptive models of the physically distributed data sources. Although these approaches provide numerous algorithms, ranging from statistical model to symbolic/logic models, they typically consider homogeneous data sites and require the support of distributed databases. As the number and size of databases and data warehouses grow at phenomenal rates, one of the main challenges in DDM is the design and implementation of system infrastructure that scales up to large, dynamic and remote data sources. The objective of our research is to equip mobile agents with system tools such that those agents can search for data sites, move from hosts to hosts, gather information and access databases, carry out complex data mining algorithms, and generate global data model or pattern through the aggregation of the local results.

To deploy mobile agents in DDM, a mobile agent system must provide languages and various programming interfaces for fast and easy development of applications. Different languages, such as C and Java, have been chosen as agent-programming languages for variety of reasons. Among them, logic-programming languages prove to be an alternative tool of building mobile agents. Benefiting from their powerful deductive abilities, complex calculations can often be represented by a set of compact logic predicates, which make agents more suitable to migrate around the network. In addition, mobile agents must interact with their hosts in order to use data services or to negotiate services with service providers. Discovering services for mobile agents came form two considerations. First, the agents possess local knowledge of the network and have a limited functionality, since only agents of limited size and complexity can efficiently migrate in a network and have little overhead. Hence specific services are required which aim at deploying mobile agents efficiently in system and network management. Secondly, mobile agents are subject to strong security restrictions, which are enforced by the system security mechanism. Thus, mobile agents should find services that help to complete security-critical tasks, other than execute code, which might jeopardize remote servers. Following this trend, it becomes increasingly important to give agents the ability of finding and making use of data services that are available in a network [3]. A variety of Service Discovery Protocols (SDPs) are currently under development by some companies and research groups. The most well known schemes are Sun's Java based Jini^{TM} [4], Salutation [5], Microsoft's UPP [6], IETF's draft Service Location Protocol [7] and OASIS UDDI [8]. Some of these SDPs are extended and applied by several mobile agent systems to solve the service discovery problem.

In a DDM environment, data may be stored among physically distributed sites and may be artificially partitioned among different sites for better scalability. Therefore endowing mobile agents with the ability of accessing remote databases is the basis for DDM applications. This encourages us to explore the strategies of coupling a mobile agent programming language with database access facilities. In recent years numerous approaches have been made under the topic of designing a coupled system that integrates a relational database and a logic programming language. They all enable programmers to access large amounts of shared data for knowledge processing, manage data efficiently as well as process data intelligently. Generally speaking, coupling a logic programming language with database access facility can be roughly divided into two categories. A loosely coupled system embeds a non-logic language, such as the Structure Query Language (SQL), within the logic-programming context and a closely coupled system provides database query interface as a subset or an extension of the language for featuring dynamic query formulation and view access. For example, SWI-Prolog [9] and KB-Prolog [10] belong to the first and CGW [11] and Quintus [12] fall into the second category.

In this paper we present the design and implementation of a mobile agent system, which couples a logic language based application programming interface for DDM. Two important system modules, namely, service discovery and database access, have been encapsulated and installed inside of the existing IMAGO (Intelligent Mobile Agent Gliding On-line) system. The IMAGO system is an infrastructure for mobile agent applications. It includes code for the IMAGO server - a Multi-threading Logic Virtual Machine, the IMAGO-Prolog - a Prolog-like programming language extended with a rich API for implementing mobile agent and DDM applications, and the IMAGO IDE, a Java-GUI based program from which users can do editing, compiling, and invoking an agent application. The remainder of the paper is organized as follows. Section 2 gives an overview of the design of service discovery module and integration with the IMAGO system. In Section 3, we discuss definitions of the database predicates and how agents to use them in DDM applications. In Section 4, we present the database connection management and briefly explain the way of integrating the database interface into the IMAGO system. Finally, we give the concluding remarks as well as future work.

2 Data Service Discovery

The general idea of distributed data services is that a DDM application may be separated from the data sites needed to fulfill a task. These data sites are mostly modeled by local and centralized databases, which are independent of, or sometimes unknown to the application. A commonly used DDM approach is to apply traditional data mining algorithms to an aggregation of data retrieved from physically distributed data sources. However, this approach may be impractical to a large scale of data sets distributed over the Internet. Deploying mobile agent paradigm in DDM offers a possible solution because the application may decompose data mining problems to scale up to a large distributed data sources and dispatch agents to carry out distributed data processing. This in turn leads us to the data service discovery problem, that is, how to find data sites available to a DDM application.

Clearly, the number of services that will become available in the Internet is expected to grow enormously. Examples are information access via the Internet, multi-media on demand, Web services and services that use computational infrastructure, such as P2P computing and Grid computing. In general, the service usage model is role-based. An entity providing services that can be utilized by other requesting entities acts as a provider. Conversely, an entity requesting the provision of a service is called a requester. To be able to offer services, a provider in turn can act as a requester making use of other services. In a distributed system, requesters and providers usually live on physically separate hosts. Providers should from time to time advertise services by broadcasting to requesters or registering their services on third party servers.

In the IMAGO system, we have implemented a new data service discovery model DSSEM (Discovery Service via Search Engine Model) for mobile agents [13]. DSSEM is based on a search engine, a global Web search tool with centralized index and fuzzy retrieval. This model especially aims at solving the database service location problem and is integrated with the IMAGO system. Data service providers manually register their services in a service discovery server. A mobile agent locates a specific service by migrating to the service discovery server and subsequently submitting requests with the required data description. The design goal of DSSEM is to provide a flexible and efficient service discovery protocol in a mobile agent based DDM environment.

Before a service can be discovered, it should make itself public. This process is called service advertisement. The work can be done when services are initialized, or every time they change their states via broadcasting to anyone who is listening. A service advertisement should consist of the service identifier, plus a simple string describing what the service is, or a set of strings for specifications and attributes.

The most significant feature of DSSEM is that we enrich the service description by using web page's URL to replace the traditional string-set service description in mobile agent systems. That is, service providers use web pages to advertise their services. Because of the specific characteristics, such as containing rich media information (text, sound, image, *etc.*), working with the standard HTTP protocol and being able to reference each other, web pages may play a key role acting as the template of the service description. On the other hand, since the search engine is a mature technology and offers an automated indexing tool that can provide a highly efficient ranking mechanism for the collected information, it is useful for acting as the directory server in our model. Of course, DSSEM also benefits from previous service discovery research in selected areas but is endowed with a new concept by combining some special features of mobile agents as well as integrating service discovery tool with agent servers.

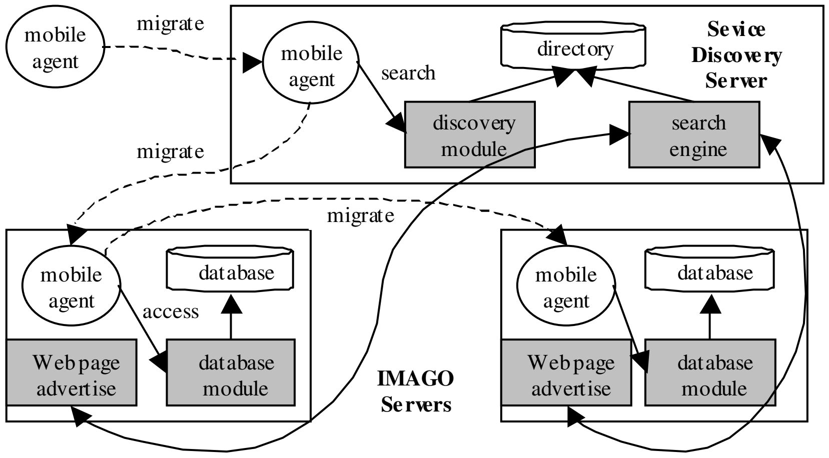

Fig. 1. An Example of Service discovery and Data Mining Process

In principle, data service providers register the URLs of their websites that advertise all the information concerning services. As a middleware on the service discovery server, the search engine will periodically retrieve web pages indicated in URLs and all their referencing documents, parse tags and words in documents and set up the relationship between the keywords and the host address of these service providers. On the other hand, a mobile agent can move to a service discovery server, utilize the system interface to access the search engine's database and obtain an itinerary that includes a list of ranked host addresses of the data service providers. Based on the given itinerary, the mobile agent may travel from host to host to carry out a DDM application. Figure 1 gives an example of service discovery and data mining process.

The application programming interface of data service discovery for mobile agents is a built-in predicate, namely, *web_search(Query, Number, Result)*, where *Query* is a compound term specifying characteristics of the search, *Number* is an integer indicating what is the maximum number of results expected, and *Result* is a variable to hold return values. For example, suppose a food company wants to analyze the customer transaction records for quickly developing successful business strategies, its DDM agent may move to a known IMAGO service discovery server and then issue a query predicate requesting up to 10 possible food industry database locations:

web_search(locate("food", "customer transaction", "imago data server"), 10, R)

The agent is blocked and control is transferred to the service discovery module of the hosting IMAGO server. The discovery module will communicate with the searcher, wait for search results, and resume the execution of the blocked agent. Search results will be delivered to the requesting agent in the form of list, where entries of the list are ranked in priorities from high to low.

3 Data Mining Facilities

Having received a list of database addresses through the service discovery module, the agent may move from host to host to access these databases or clone multiple agents with assigned database addresses to start the DDM application. In order to bridge logic based mobile agents with database systems, the IMAGO system provides a set of database predicates, which enables agents to establish connection with data sources and make requests for desired information.

A database connection is established by issuing a predicate of the form *db_connection(Info , ID , E)*, where the input argument *Info* binds with the address and authorization information, such as database address, user name and password, the output argument *ID* will be bound with an integer uniquely identifying a successful connection, and *E* will be bound with an integer, either 0 or an negative error code indicating if the current predicate succeeds or fails. To disconnect a database, the agent may issue a predicate *db_disconnection(ID , E)*, where *ID* represents the connection to be closed, and *E* returns execution result. Suppose that an agent has obtained a public-accessible database server address *xxx.yyy.zz.ww*, and it wants to access data anonymously, the agent may simply execute the following code:

```
Info = connect("xxx.yyy.zz.ww", "anonymous", _),
db_connection(Info, ID, E1),
// Access and apply data mining algorithm through connection ID
db_disconnection(ID, E2).
```

To facilitate DDM applications, the IMAGO system provides two different ways of database access operations, *i.e.*, the set retrieval and the tuple retrieval. The former returns the entire matching data set to the requesting agent, whereas the latter allows the requesting agent to consume the matching data on the tuple by tuple basis. A set retrieval operation is defined by predicate *db_search_set(ID , DB , Q , Dataset , E)*, where *ID* indicates a pre-established database server connection, *DB* is bound with a string representing the target database name, *Q* gives the searching query, *Dataset* is a variable holding the retrieval result, and *E* is a variable to be bound with a positive integer, which gives the number of records in the data set if the operation succeeds, or a negative integer if the operation fails. For example, suppose that an agent has established a database server connection *ID*, it can search for the whole set of customer records from a public database by issuing the following predicates:

*db_search_set(ID, "public_database", "select * from customer", Dataset, E),*
data_mining_algorithm(Dataset).

Obviously, the set retrieval is not practical if the return result involves a huge amount of matching records. The operation of tuple retrieval is introduced to handle this case. There are two predicates to implement tuple retrieval operations, *i.e.*, *db_search_tuple* and *db_tuple_next*. The first predicate shares the same syntactic form as the set retrieval predicate. Its behavior is to initialize a system manipulated searching buffer and return the first marching record. The second predicate consumes the next available data record from the system buffer orderly. Remaining records are maintained by the IMAGO database module and linked with the connection identified by *ID*. Therefore through this linkage the matching data records can be sequentially consumed by a sequence of *db_tuple_next* predicates, which are probably invoked recursively. An example of using tuple retrieval is given by the following example:

*db_search_tuple(ID, "public_database", "select * from customer", DataRec, E),*
data_mining_algorithm(ID, DataRec), // invoke data mining algorithm

where the procedure *data_mining_algorithm* may be defined as follows:

```
data_mining_algorithm(ID, DataRec):-
        // Processing the current DataRec,
        db_tuple_next(ID, DataRecNext, E),
        E<0 →
         terminate;
         data_mining_algorithm(ID, DataRecNext).
```

Through the recursive invocation of *db_tuple_next* predicate, the entire result set is accessed on the record-by-record basis. The value of *E* in the *db_tuple_next* predicate can be used as a recursion termination condition because it will be bound with a negative integer when the bottom of the record set is reached.

In a DDM application, agents are not working alone and they need to communicate with each other to cooperate and generate a global data aggregation for further analysis. Most existing mobile agent systems adopt some kind of communication models/ protocols from traditional distributed systems. However, the IMAGO system adopts a different strategy to cope with this issue. The idea is to deploy intelligent mobile messengers for inter-agent communication [14]. Messengers are thin agents dedicated to deliver messages. Like normal agents, a messenger can move, clone, and make decisions. Unlike normal agents, a messenger is anonymous and its special task is to track down the receiving agent and reliably deliver messages in a dynamic, changing environment. The IMAGO system provides a set of built-in messengers as a part of its programming interface. A data-mining agent at any remote sites and at any time may dispatch messengers to deliver data to designated receivers. For example, suppose that a mobile agent has completed its data mining work at a remote database server, it can either call *move('home')* to carry results and migrate back to its stationary server, or invoke *dispatch($oneway_messenger, 'home', Results)* to create a messenger which is responsible to deliver *Results* to the *home* server.

Communication among agents takes place by means of an Agent Communication Language (ACL). The essence of an ACL is to make data mining agents understanding the purpose and meaning of their exchanged data. In general, a message consists of two aspects, namely, performative and content. The performative specifies the purpose of a message and the content gives a concrete description for achieving the purpose. In order to facilitate open standards of ACL's, the IMAGO agent-based communication model is in compliance with the FIPA ACL message structure specification [15]. Of course, the performative and content of a message often determine the reaction of the receiver. In addition to the various types of system built-in messengers for sending agents, the IMAGO system provides a set of predicates for receiving agents. The predicate which is similar to an unblocking receive is *accept(Sender, Msg)*. An invocation to this procedure succeeds if a matching messenger is found, or fails if either the caller's messenger queue is empty or there is no matching messenger in the queue. Likewise, the predicate which implements the concept of blocking receive is *wait_accept(Sender, Msg)*. A call to this procedure succeeds immediately if a matching messenger is found. However, it will cause the caller to be blocked if either the caller's messenger queue is empty, or no matching messenger can be found. In this case, it will be automatically re-executed when a new messenger attaches to the caller's messenger queue. Pragmatically, the semantics of matching messengers is implemented by logic unification.

4 Database Connection Management

In the design of a logic based DDM system, an important consideration is efficiency. Several proposals on efficiently manipulating databases have been made at both technical level and logic level. On the technical level, a database connection management scheme is proposed to reduce the cost of network establishments by reusing existing database connections [16]. In addition, the data set retrieved from databases can be handled efficiently by a cache management system, which stores pre-fetched data and enables most queries to be fulfilled in these local caches at least partially [17]. On the logic level, a method of delayed evaluation is introduced by [18], which prevents redundant information being retrieved in a single logic program context. Obviously, establishing, maintaining, or terminating a database connection is always regarded as an expensive job, which not only occupies network resources but also takes significant amount of user/system overhead. The idea of connection management is to let all database operations issued by agents share a number of pre-generated database connections, and therefore many unnecessary establishment and termination operations can be avoided.

To simply our discussion, let *C* denote a set of database connections, where each entry of *C* is a quadruple *(Id, Status, Name, Link)*. As before, *ID* is an integer identifying a connection, *Status* holds the current connection states, such as *FREE, CONNECTED*, or *OCCUPIED*, *Name* gives the symbolic name of the database and *Link* indicates the address of the database associated with the connection. As an example, let us assume the following set of database connections:

C = [(1, FREE, _, NULL), (2, CONNECTED, "public_database", "131.104.49.49"), (3, OCCUPIED, "localhost", "127.0.0.1")].

It is clear that the first entry of *C* represents a *FREE* connection, the second entry holds a *CONNECTED* state indicating that the connection has been established and linked to a target database but is not being used currently, whereas the last entry shows an *OCCUPIED* connection, which is connected to the database installed on the *localhost* and an agent is accessing data through this connection. Initially all entries in *C* are *FREE*. Whenever an agent issues a connection request, the IMAGO database module will first try to find a *CONNECTED* entry with the condition that whose target database is the same as that requested by the *db_connection* predicate. The selected connection is ready for data retrieval because it has already connected to the desired database. However, if such an entry cannot be found, a *FREE* entry will be picked up and connection establishment operation is invoked. A worse case is that set *C* has reached its maximum capacity and none of the above two types of entries exist. Therefore, the system must find a *CONNECTED* entry, disconnect it, and set it *FREE* to establish a new connection to the required database. Of course, the state of a selected entry needs to be changed from either *CONNECTED* or *FREE* to *OCCUPIED* before being used and a new pair of username and password provided by the *db_connection* predicate needs, if necessary, to be re-authorized by the database system. Finally, if all entries of *C* are of *OCCUPIED* state, the subsequent connection requests will be blocked until some data mining agents release their connections explicitly.

Having finished the desired data mining, an agent may call *db_disconnection* predicate to return the database connection by simply changing its state from *OCCUPIED* to *CONNECTED* without actually terminating the connection with the target database. Other agents would probably reuse this non-terminated connection sooner or later and therefore save network resources and reduce system/user overhead.

5 Conclusion

In this paper, we discussed the scheme of deploying mobile agents in DDM applications. The advantage of adopting mobile agents for DDM is to scale up to large, dynamic and remote data sources, such as various databases distributed over the Internet. We presented the design of data service discovery module and database management module. The programming interface of these modules is a set system built-in predicates capable to couple a logic programming language with functionalities of locating data services and accessing remote databases. Equipped with those system tools, mobile agents may search for suitable data sites, roam the Internet to collect useful information, and communicate with each other to generate a global view of data through the aggregation of distributed computations. In order to verify the feasibility and efficiency of the mobile agent based DDM proposal, experimental service discovery module and database management module have been implemented and integrated with the IMAGO system. The service discovery module is based on the search engine technology and concentrates on locating database services. It uses web pages as a medium to advertise services, and runs an independent search engine to gather and index service provider's information, such as service types, database names, URLs, access modes, as well as possible verification information. The database management module not only provides flexible interface for accessing data, but also manipulates database connections efficiently. At the current stage, the database model in the IMAGO system is MySQL, the most popular open source DBMS system in the world [19].

Research on the agent based DDM involves further extensions of the IMAGO system. First, the current implementation of service discovery module deals with only a limited number of logical relationships. To be able to offer more precise discovery service, this module could be enhanced to parse some complex search criteria, such as conditional expressions and sub-string matching. Secondly, since databases may contain multi-dimensional data, retrieving such kind of information from flat web pages is a pending problem. We are looking to use XML meta-data to solve the database dimensional problem. In addition, we are making investigations on adding more programming languages to the IMAGO system, as well as introducing more flexible and efficient communication tools, such as mobile socket, to facilitate DDM applications.

Acknowledgments. We would like to express our appreciation to the Natural Science and Engineering Council of Canada and State Key Laboratory of Novel Software Technology (Nanjing University) for supporting this research.

References

[1] Klusch, M., Lodi, S., Moro, G.: The Role of Agents in Distributed Data Mining: Issues and Benefits. In: IEEE/WIC International Conference on Intelligent Agent Technology (IAT 2003) (2003)

[2] Park, B., Kargupta, H.: Distributed Data Mining: Algorithms, Systems, and Applications, Data Mining Handbook. In: Ye, N. (ed.) (2002)

[3] Bettstetter, C., Renner, C.: A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol. In: Proc. of EUNICE 2000, sixth EUNICE Open European Summer School, pp. 13–15 (2000)

[4] Hashman, S., Knudsen, S.: The Application of Jini Technology to Enhance the Delivery of Mobile Services, White Paper (2001), http://wwws.sun.com/

[5] Salutation Consortium, Salutation Architecture Overview, White Paper (1998), www.salutation.org/whitepaper

[6] Universal Plug and Play Forum, Universal Plug and Play Device Architecture, Version 0.91, White Paper (2000)

[7] Guttman, E., Perkins, C.: Veizades, Service Location Protocol, Version 2, White Paper, IETF, RFC 2608 (1999)

[8] OASIS UDDI, UDDI White Paper (2005), http://www.uddi.org

[9] Wielemaker, J.: SWI-Prolog ODBC Interface, technical report University of Amsterdam, The Netherlands (2000)

[10] Bocca, J., Dahmen, M., Macartney, G.: KB-Prolog User Guide, Technical Report, ECRC Munich (1989)

[11] Ceri, S., Gottlob, G., Wiederhold, G.: Efficient Database Access from Prolog. IEEE Transactions on Software Engineering 15(2), 153–164 (1989)

[12] Quintus Inc., Quintus Prolog Database Interface Manual, technical report (1998), http://www.sics.se/

[13] Song, L., Li, X., Ni, J.: A Database Service Discovery Model for Mobile Agents. International Journal of Intelligent Information Technologies 2(2), 16–29 (2006)

[14] Li, X., Autran, G.: Inter-agent Communication in IMAGO Prolog, Lecture Notes in Artificial Intelligence. In: Bordini, R.H., Dastani, M., Dix, J., Seghrouchni, A.E.F. (eds.) Programming Multi-Agent Systems. LNCS (LNAI), vol. 3346, pp. 163–180. Springer, Heidelberg (2005)
[15] FIPA ACL, Agent Communication Language Specifications, FIPA (2005), http://www.fipa.org
[16] Mckay, D., Finin, T., O'Hare, A.: The Intelligent Database Interface: Integrating AI and Database Systems. In: Proceedings of the 8th National Conference on Artificial Intelligence, pp. 677–684 (1990)
[17] Sheth, A., O'Hare, A.: The Architecture of BrAID: A System for Bridging AI/DB Systems. In: Proceedings of the Seventh International Conference on Data Engineering, pp. 570–581 (1991)
[18] Cuppens, F., Demolombe, R.: A Prolog-relational DBMS Interface Using Delayed Evaluation. In: Proc. 3rd Int. Conf. On Data and Knowledge Bases, pp. 135–148 (1988)
[19] MySQL AB MySQL Reference Manual, user's manual (2005), http://dev.mysql.com/doc/mysql/en/