A Scalable and Robust Framework for Agent-Based Heterogeneous Database Operation

Chunsheng Li\(^1\), Chengqi Zhang\(^1\), Qingfeng Chen\(^1\), and Zhih Zhang\(^2\)
\(^1\) Faculty of Information Technology, University of Technology, Sydney
PO Box 123, BROADWAY, NSW 2007, Australia
E-mail: \{csl, chengqi, qchen\}@it.uts.edu.au
\(^2\) School of Information Technology, Deakin University, Geelong
Victoria 3217, Australia
E-mail: zzhang@deakin.edu.au

Abstract

How to operate database efficiently and unfailingly in agent-based heterogeneous data source environment is becoming a big problem. In this paper, we contribute a framework and develop a couple of agent-oriented matchmakers with logical ring organization structure to match task agents' requests with middleware agents of databases. The middleware agent is a wrapper of a specific database and is run on the same server with the database management system. The matchmaker is of the features of proliferation and self-cancellation according to the sensory input from its environment. The ring-based coordination mechanism of matchmakers is designed. Two kinds of matchmakers, namely, host and duplicate, are designed for improving the scalability and robustness of agent-based system. The middleware agents are improved for satisfying the framework. We demonstrate the potentials of the framework by case study and present theoretical and empirical evidence that our approach is scalable and robust.

1 Introduction

With the agent-based technique becoming the mainstream of software engineering paradigm, many agents must be efficiently organized for their interaction in a good situation [1]. The interaction among agents is dependent on the organizational structure and coordination mechanism. Organizational structure presents the interrelationship among agents in a system, and coordination mechanism is knowledge level protocol to control the sequences of interaction and manage conflicts among agents. According to the role of each agent or interrelationship among agents, agent-based system has an organizational structure. The organizational structures have been modeled into three types, namely, Peer-to-Peer, Tree, and Grouping with facilitator, in practical agent-based systems. Coordination mechanism is the protocols to manage interdependencies between the activities of agents. Coordination mechanisms have been classified into four patterns, namely, Direct search, Matchmaker, Broker, and Contract-net [2, 3].

Today a lot of databases that are usually heterogeneous and store gigabytes of data are used by different systems in many areas. Users and developers of new complex problem solving systems which operate those databases all hope to inherit legacy data with the novel agent-based method and do not like to affect the current running systems which may be the “owners” of those databases. Certainly legacy data can be migrated from current databases to a new agent-based database whose management system must be developed with the agent perspective. But it cannot be imagined to migrate synchronously a legacy relational database system to the agent-based platform. For operating databases in agent-based environment, we employed broker coordination mechanism to wrap the database with an agent that is called middleware agent. The middleware agent is a wrapper of a specific database and is run on the same server with the
database management system. From agent point of view, the middleware agent makes database exhibit the agent characteristics. Task agents each of which completes a specific task operate databases by means of those middleware agents [4].

We employed matchmaker coordination mechanism between task agents and middleware agents [4] for coordinating their interactions. But by using the current valid architectural model F-M, which combines Group with facilitator (F) and Matchmaker (M), there are two limitations. Firstly, the matchmaker is becoming the bottleneck between agent interactions of the system. When a task agent wants to interact with a middleware agent, it must contact with matchmaker first. If the number of task agent is enormous, the matchmaker will be very busy. At last, because of the overload of the matchmaker, the system will be broken down or the efficiency of the system will decrease to a very low point. Secondly, because the reliability of the system absolutely relies on the matchmaker, the robustness of the matchmaker is the key problem. If the matchmaker crashed, the whole system would break down. Certainly, the above limitations can be abated by means of increasing amount of matchmaker. Because of the lack of efficient organization structure, the limitations cannot be eliminated at all. In this paper, we contribute a framework and develop a couple of agent-oriented matchmakers with logical ring organization structure to match task agents’ requests with middleware agents of heterogeneous databases. The matchmaker is of the features of proliferation and self-cancellation according to the sensory input from its environment. The ring-based mechanism of matchmakers is designed. Two kinds of matchmakers, namely, host and duplicate, are designed for improving the scalability and robustness of the framework.

The remaining sections of this paper are organized as follows: Section 2 chiefly presents the related work about this research. Section 3 introduces the methodology of ring-based matchmaker architectural model. Section 4 describes the scalable and robust framework for agent-based heterogeneous database operation by means of middleware agents and ring-based matchmakers. Section 5 discusses the architecture of middleware agent. Section 6 gives a case study and evaluation of the framework. Section 7 concludes the paper.

2 Related work

To authors’ knowledge, such an architectural model of ring-based matchmakers for heterogeneous database operations is new. However, the related technologies or methodologies employed in this paper are becoming very hot topics in the field of intelligent agents [1] and distributed systems [5] at the moment. Because there is insufficient space here to cover the gamut of work on agent, distributed system and database operation related research, we restrict ourselves to mentioning several projects that have helped to develop our framework.

How to design architecture is very important issue in agent-based system construction. Because agent-based system achieves its goal through interaction among agents, it is very important to design agent-based system so that agents can efficiently interact with each other. Managing the interaction is the goal of coordination: the space of agent interaction is no longer to be seen as merely the space of communication, but also the space of coordination [6]. Lee and Kim researched architecture modeling and evaluation for generic agent-based systems. They proposed seven valid architectural models for agent-based systems by combining three organization structures (Peer-to-Peer, Grouping with facilitator, and Tree) [2, 3, 7] and four coordination mechanisms (Direct search, Matchmaker, Broker, and Contact-net) [3, 7]. The idea of this research is very important for designing agent-based system. However, the models are not enough to cover the gamut of work on agent-based system construction, especially, on the
scalability and robustness of agent-based systems. Jelasity, et al. proposed a novel tool based on a scalable and robust framework for running distributed applications [8]. The coordination mechanism is based on cutting-edge peer-to-peer technology. Papastavrou, et al. proposed a framework for Web-based distributed access to database systems based on Java-based mobile agents [9]. In DBMS-aglet system, mobile agent (DBMS-aglet) was employed. The system lost some benefit for transferring DBMS-aglet between client and servers. Li, et al. contributed a framework and developed an agent-based middleware with legacy software wrapping view to operate heterogeneous databases in uniform database-based agent communication language (DBACL) [4]. In terms of the framework architecture, to cooperate all middleware agents is a big problem. Tewissen, et al. presented a software toolkit ‘matchmaker’ allowed for synchronizing already existing software applications by plugging in transparent event listener mechanisms that distribute information via a central server to other remote applications [10]. All above work somewhat promoted our research.

3 Methodology

3.1 Architecture modeling for matchmaker

Architectural model for agent-based system is based on organization structure and coordination mechanism. Ring-based architectural model is powerful for improving the scalability and robustness of the key components of agent-based system. The ring-based architectural model is based on logical ring organization structure and token ring coordination mechanism. The logical ring organization structure is showed in Figure 1.

![Figure 1 Ring organization structure](image)

The facilitator A, B, C, and D control other agents, that is, agent group 1 (AG1) and agent group 2 (AG2). Each agent group includes a set of agents that are of similar features or close cooperate to achieve a chief goal. The agents in an agent group may be organized as peer-to-peer or tree structure. An agent in one agent group cannot directly contact with one in another agent group. The facilitators are organized as logical ring structure. That is, facilitator A can only contact directly with facilitator B and D, but C. If facilitator A wants to contact with C, the message must be transmitted by B or D. For simplifying control mechanism, the logical ring is designed to be of the feature of one way in transmitting message. For example, facilitator A can only directly transmit message to facilitator B and receive message from D, but it is not true in contrast. Each facilitator stores meta necessary information (like yellow page) about its supervised child agents to control the interaction among those child agents. The matchmakers are divided as two categories, namely, host and duplicate, according to their roles in the system. The host facilitators, for example A and C, are the superior agents of agent group 1 and agent group 2, respectively, in normal case. If the host facilitator A or C
crashed, the ring would be automatically reorganized. At the same time, its duplicate facilitator D or B will act as a host and automatically proliferate a new duplicate facilitator. The meta information in host facilitator and its duplicate must be synchronized in time.

3.2 Matchmaker coordination mechanism

Coordination mechanism is another factor related to the interaction and organization among agents and facilitators. Without coordination, agents will not work in order because of the conflicts between them and cannot correctly respond to service requests. The coordination mechanism of matchmaker is to match task agents’ requests with the middleware agents and makes the middleware agents be able to answer to the requesters. The token ring coordination mechanism of matchmaker, which consists of logical ring establishing and token control system, is designed in this research.

For efficiently establishing the logical ring of matchmakers, there is a coordinator among matchmakers at any time to establish a logical ring (system beginning, token lost or the ring broken down). In general, it does not matter which matchmaker takes on this special responsibility, but one of them has to do it. For selecting a matchmaker as coordinator, each matchmaker is assigned a unique number (0 to 255). In general, election algorithm attempts to locate the matchmaker with the highest matchmaker number and designate it as coordinator. Furthermore, it is assumed that every matchmaker knows the matchmaker number of every other matchmaker. What the matchmakers do not know is which ones are currently up and which ones are currently down. The goal of election algorithm is to ensure that when an election starts, it concludes with all matchmakers agreeing on who the new coordinator is to be. In our research, we employed ring election algorithm [5]. This algorithm is based on the use of a ring, but without a token. It is assumed that the matchmakers are logically ordered, so that each matchmaker knows whom its successor is. When any matchmaker notices that the coordinator is not functioning (the token interval is overtime), it builds an ELECTION message containing its own matchmaker number and sends the message to its successor. If the successor is down, the sender skips over the successor and goes to the next member along the ring, or the one after that, until a running matchmaker is located. At each step, the sender adds its own matchmaker number to the list in the message. Eventually, the message gets back to the matchmaker that started it all. That matchmaker recognizes this event when it receives an incoming message containing its own matchmaker number. At that point, the message type is changed to COORDINATOR and circulated once again, this time to inform everyone else who the coordinator is (the list member with the highest number) and who the members of the new ring are. When this message has circulated once, it is removed and coordinator starts to manage the ring. The coordinator is in charge of managing the token, receiving middleware agent’s advertisement, receiving new system request for proliferating a pair of new matchmakers (host and its duplicate) for this new system, receiving a system removing request for canceling the related matchmakers (host and its duplicate) and removing them from the ring, maintaining he logical ring (control to proliferate new matchmakers or remove old ones), etc.

When the ring is initialized, the coordinator will produce a token. The token circulates around the ring. It is passed from matchmaker k to matchmaker k+1 (modulo the ring size, 256) in point-to-point messages. When a matchmaker acquires the token from its neighbor, it has three aspects of tasks to do. Firstly, matchmaker checks to see if there is middleware agent attempting to advertise it to matchmaker. If yes, the matchmaker registers middleware
information. Secondly, if there is any maintaining message (new system information, canceled system information, applied matchmaker numbers, etc.) on the token, the matchmaker updates related items or statuses. Thirdly, if the matchmaker is a host, it checks if its duplicate is normal (If not, it proliferates a duplicate and makes it on the ring). If the matchmaker is a duplicate, it checks if its host is normal (If not, it acts as the host, proliferates a duplicate and makes it on the ring). When the coordinator holds the token, it will clear the token and reload it again according to the environment.

3.3 Architecture of the matchmaker

Matchmaker is of four aspects of tasks. Firstly, it matches task agent’s request with a suitable middleware agent. Secondly, it may act as a coordinator. Thirdly, it looks after its duplicate or vice versa. The last, it receives and transmits the token on the logical ring. For achieving above tasks, a matchmaker consists of eight components: Communication Protocol Module (G PORT and C PORT), Coordinator Processing, Ring Manager, Token Manager, Middleware Agent Table, Status indicators, Matching, Control Module. The architecture of matchmaker is shown in Figure 2.

![Architecture of the matchmaker](image)

Communication Protocol Module deals with the interaction with other agents by means of Database-Based Agent Communication Language (DBACL) [6]. In the lowest tier, there are two communication ports, one for General purpose (G PORT) and another for Coordinator purpose (C PORT). All matchmakers use the same port number to C PORT for convenient registration of new systems and middleware agents, but just one matchmaker is available at any time. If the matchmaker acts as a coordinator, the C PORT is available; otherwise it is prohibited. Coordinator Enabling maintains this function. Coordinator Processing deals with the coordinator election, generating a token when the ring is initialized, managing current systems, proliferating a pair of new matchmakers for the new system, removing a pair of matchmakers for the canceled system, and managing the registration of middleware agent. When coordinator holds the token, it puts the registration information of middleware agent in the token. Other matchmakers will get the information from token and save it to the Middleware Agent Table. If a middleware agent is cancelled, the related item will be removed from the Middleware Agent Table. Ring Manager deals with changing status of the matchmaker (from duplicate to host), proliferating a duplicate and adding it on the ring. There are two indicators to indicate its successor and predecessor on the ring. Token Manager deals with the receiving token, doing all tasks when a matchmaker holds the token as described in Subsection 3.2, updating Middleware Agent Table and Status indicators in accordance with the information on the token, transmitting the token to its successor. Middleware Agent Table
holds the information about middleware agents. Status indicators include matchmaker status indicator, current system indicator, and the list of allocated matchmaker numbers. Matching Module matches task agent’s request with a suitable middleware agent. Control Module, which is the control center of the matchmaker, makes other components work in order.

4 Framework for database operation

The matchmakers with ring-based architectural model are applied in the heterogeneous database operation systems. A pair of matchmakers (host and its duplicate) is corresponding with a specialized agent-based system that may be a subsystem of a complex system. There are not direct interactions between these subsystems. The interactions are implied in the data in databases or data files. In this framework, we regard subsystem as system.

4.1 Framework with ring-based matchmakers

The framework consists of three parts, that is, system-oriented task agents ($A_{11}$, $A_{12}$, ..., $A_{1n}$, $A_{21}$, $A_{22}$, ..., $A_{2n}$), ring-based matchmakers (A, B, C, and D. Assumed that D is the coordinator), and middleware agents with their databases ($M_1$ with $DB_1$, $M_2$ with $DB_2$, ..., $M_x$ with $DB_x$). Ring-based matchmakers are placed in the middle between task agents of systems and middleware agents of databases. The framework is shown in Figure 3.

![Figure 3 Framework for database operation](image)

The relationships between task agents (Peer-to-Peer or Tree) in one system are not described in Figure 3. We assumed that there are two independent systems to contact with middleware agents. System 1 consists of task agent $A_{11}$, $A_{12}$, ..., and $A_{1n}$. System 2 consists of task agent $A_{21}$, $A_{22}$, ..., and $A_{2n}$. Host matchmaker A directly serves for system 1. Matchmaker D is the duplicate of matchmaker A and acts as the coordinator at the moment. Host matchmaker C directly serves for system 2. Matchmaker B is the duplicate of matchmaker C. The part of middleware agents includes database management systems and middleware agents that are run in one server or more, but each middleware agent only directly operates a specific database. If one task agent wants to operate any database, it has to communicate with the database’s middleware agent by means of DBACL. Although the databases are heterogeneous, to operate the databases is uniform by means of middleware agents with DBACL.
4.2 Interaction between agents by matchmakers

Matchmakers automatically construct themselves to satisfy the environment of the middleware agents and current systems. At the same time, they coordinate the interactions between task agent and middleware agent. The interactions can be divided into three categories: managing middleware agent registration, managing current system registration, and matching task agent with middleware agent. Figure 4 presents the interaction patterns.

![Diagram showing interaction between agents](image)

Figure 4 Interaction between agents

When a middleware agent wants to register for providing service to task agents, it advertises its capability and feature to the matchmaker which is acting as the coordinator (A1 in Figure 4). When the coordinator holds the token, it registers the middleware information into Middleware Agent Table, puts the information on the token, and informs all other matchmakers to register the middleware information as it did. When the token comes back, coordinator clears this information from token. When a middleware doesn’t want to provide service to task agent, it sends ‘unadvertise’ to the coordinator (A2 in Figure 4). When the coordinator holds the token, it removes the middleware information from Middleware Agent Table, puts the information on the token, and informs all other matchmakers to remove the middleware information as it did. When the token comes back, coordinator clears this information from token. If a matchmaker detects a crashed middleware agent, it will ask coordinator to do the similar work as ‘unadvertise’.

When a new system wants to register for constructing its matchmakers (host and duplicate), it sends its information to the coordinator (B1 in Figure 4). When the coordinator holds the token, it registers the new system’s information into Current System List, proliferates two matchmakers for the system and makes them on the ring (if there are not enough vacant matchmaker numbers, cancel all those actions), puts the information on the token, and informs all other matchmakers to register the system information as it did. When the token comes back, coordinator clears this information from token and replies the new system with related matchmaker’s access port number (B2 in Figure 4). The system informs all its task agents of the matchmaker’s port number. When a system doesn’t want to run forever, it sends ‘remove’ to the coordinator. When the coordinator holds the token, it removes the related matchmakers from the ring (if there are only two matchmakers on the ring, this step will be overlapped), removes the system information from Current System List, puts the information on the token, and informs all other matchmakers to remove the system information as it did. When the token comes back, coordinator clears this information from token.
For matching task agent with middleware agent, the steps are as follows.
(1) Task agent asks its related matchmaker to answer to its request (C1 in Figure 4).
(2) Matchmaker searches all middleware agents in middleware agent table, selects a middleware agent which is able to answer to the request, and replies the middleware agent’s information to task agent (C2 in Figure 4).
(3) Since task agent knows which middleware agent is able to solve its request, it directly asks the middleware agent to answer the request (C3 in Figure 4).
(4) The middleware agent completes the request and replies the result to the task agent (C4 in Figure 4).

After matching task agent with middleware agent, matchmaker does not intervene any interaction between task agent and middleware agent, that is, Step 3 and 4 can be repeated for many times for other similar tasks.

5 Architecture of the middleware agent

Because the databases are of different characteristics, the middleware agents for database operation have a special architecture. In this section, the characteristics of database operation are firstly introduced. And then contribute the architecture of middleware agent.

5.1 Characteristics of database operation

Databases are different from other kinds of non-agent software systems. Firstly, the main objective of database is to organize, store, and manage neuter data because database management systems care naught for the meaning of the data but their data types. Secondly, the direct users of database are usually other application programs by means of database operation language (e.g. SQL) or application programming interface (API, e.g. ODBC, JDBC). Thirdly, database management systems in a specialized application usually run in stationary servers and cannot be transferred randomly. Fourthly, many database management systems generally provide more than one developing environment for designers’ convenience to develop their diversiform application systems. For example, Oracle not only provides SQL functions to operate the database in database programming environment, but also Pro*C to develop application systems in C programming language environment. For the above characteristics of databases, we know that databases are of close feature. Because of the close feature, it is reasonable to wrap database with an agent wrapper [4].

5.2 Architecture of the agent-based middleware

The middleware agent consists of Socket Module, DBACL Checking Module, Advertise Module, Task Executing Module, Database Operating Module, Output Results Generation Module, Knowledge For Database Operation Module, and Control Module. The architecture is shown in Figure 5.

![Architecture of the middleware agent](image)

Figure 5 Architecture of the middleware agent
Socket Module deals with the communication with other agents. Socket is a formalization objective between two ports of communicating programs. This part is the lower layer of DBACL and makes the DBACL formats satisfy Socket and vice versa. DBACL Checking Module deals with parsing of DBACL statement according to the rules and database information stored in the module of Knowledge For Database Operation Module through Control Module. If the DBACL statement is valid, it is transmitted to Task Executing Module, otherwise transmitted to Output Results Generation Module to generate an error message as response. Advertise Module deals with registering or removing this middleware agent to/from coordinator. When the middleware agent is initialized, the Control Module tells Advertise Module to register the capability and feature of the middleware agent. The module advertises to coordinator by Output Results Generation Module. When database is shut down, the Control Module tells Advertise Module to remove the information of the middleware agent by coordinator. Task Executing Module decompounds the DBACL statements as SQL statements to operate database through Database Operation Module. On the other hand, this module organizes the results from database and transmits them to Output Results Generation Module. All function of this module complies with the knowledge stored in Knowledge For Database Operation Module. Database Operating Module is the interface to operate the database. It is a Pro*C program module with dynamic embedded SQL technique. Output Results Generation Module forms DBACL statements based on the rules stored in Knowledge For Database Operation Module. The inputs of the module are from DBACL Checking Module or Task Executing Module and the outputs are DBACL statements with the results accessed database or with responses. Knowledge For Database Operation Module manages and stores the DBACL rules and database operation information. Control Module, which is the control center of the middleware agent, makes other modules work in order.

6 Case study and evaluation

The framework and its implementation described in this paper have been applied in an industrial project----An agent-based curve-digitizing approach to well-logging data manipulation [11]. For evaluating the novel framework, two systems, namely, curve-digitizing system and integrated data query system, are designed. The agent-based systems, matchmakers, and middleware agents worked well by processing test with thousands of blueprints.

The goal of curve-digitizing system is that the well-logging curves on blueprint are digitized and the data are stored in database. The tasks of the system are as follows. (1) The blueprint with well-logging curves is scanned as an image file (TIFF format) by scanner. (2) The image file is preprocessed as the standard pattern file and then compressed to characteristic data files, which are saved to data file servers. (3) The curves implied in those data files are digitized to curve data. The curve data are stored into Oracle database. In accordance with above tasks, three task agents, namely, interface agent, preprocessing agent, and curve-digitizing agent, are designed. Interface agent interacts with user, for example, inputting text or data by keyboard, scanning image into computer by scanner, controlling to execute specialized function, or displaying statuses of the system. This agent is of the scanning function. It completes the scanning task and saves the results as temporary image files. Another task of this agent is to contact with the coordinator for registering or removing the system and to manage the matchmaker information for the system (e.g. it informs all task agents in the system about the matchmaker information). Some images may need to be preprocessed for forming a standard image pattern before compressing and digitizing.
Preprocessing agent completes above preprocessing task. The agent compresses standard image pattern files to data files, which will be stored forever in hard disks. At the same time, a large amount of hard disk space is saved. Curve-digitizing agent digitizes the compressed image files, which include skeleton type of curves and implicate some data in those curves (Here, skeleton type of curve means that the shape of the curve is only preferred, but the width of the curve is not important). Because some kinds of curves are ambiguous in digitizing process, digitizing knowledge is required in order to getting accurate data. This agent also deals with the interactions with related matchmaker and middleware agent (The matchmaker information is from interface agent of the system).

The goal of integrated data query system is that the data or files in database or file server can be queried in text or image modes. The tasks of the system are as follows. (1) The curve data or other information are queried from databases in data mode or image mode. (2) The characteristic data files are queried in image mode. In accordance with above tasks, three task agents, namely, interface agent, data query agent, and curve redrawing agent, are designed. Because the system must interact with users, for example, querying curve data displayed in text, querying regenerated images that include well-logging curves, controlling to execute specific function, or displaying statuses of the system, Interface agent deals with interaction between users and the system to achieve above tasks. Another task of this agent is to contact with the coordinator for registering or removing the system and to manage the matchmaker information for the system (e.g. it informs all task agents in the system about the matchmaker information). Data query agent deals with accessing and processing curve data or other information in Oracle database or data file servers by means of matchmaker and middleware agent (The matchmaker information is from interface agent of the system). Because data can be displayed in text mode or image mode, the data, which are from database or data files, need to be processed before they are displayed. Curve redrawing agent gets curve data or other information from data query agent and redraw the blueprint again.

The architecture model and its behaviour are related to the performance of agent-based system. We evaluate the ring-based architecture model with performance predictability, adaptability, and availability. The performance predictability can be measured by complexity and efficiency of the system. Adaptability is measured by extendibility of the system. Complexity is defined as the number of links among agents. Efficiency is defined as the number of links from service request to completion of the service. Extendibility is defined as the number of links that need to add agent or facilitator. Availability is defined as following:

\[ \text{Availability} = \frac{(T-F)}{T} \]

Where: T = Total number of links; F = Number of links connected to abnormal agent or facilitator.

The results for the case study above are as following (Assume that two middleware agents are used in the case): Complexity is 22, which is worse than Tree (less than 20) but better than Peer-to-Peer and Grouping with facilitator (between 30 to 70). Efficiency is 14, which is better than Tree, Peer-to-Peer and Grouping with facilitator (Between 30 to 50). Extendibility is 4, which is worse than Tree (2) but better than Peer-to-Peer and Grouping with facilitator (7, 12 respectively). Availability is 0.82, which is similar with Tree, Peer-to-Peer and Grouping with facilitator (Between 0.8 to 0.9). (Note that all above comparisons are based on Lee and Kim's work at Korea University).

We would like to compare the framework of this paper with DBMS-aglet system proposed by Papastavrou, et al. [9] to show the benefit of the middleware agent. In DBMS-aglet system, mobile agent (DBMS-aglet) was employed. The system lost some benefit for
transferring DBMS-aglet between client and servers. Because the middleware agent is stationary, it outperforms DBMS-aglet in benefit and flexibility. However, the DBMS-aglet system promoted a much more efficient way of utilizing the JDBC (Java DataBase Connectivity) and eliminated the overheads of the various conventional approaches. In wireless and dial-up environments and for average size transactions, a client/agent/server adaptation of the DBMS-aglet system provided a performance improvement of approximately a factor of ten. For the fixed network, the gains were about 40 percent [9].

7 Conclusion

To promote the scalability and robustness of matchmaker in agent-based system, we introduced a ring-based architecture model. This model enriches the architecture of facilitator that is the distillation of most agent-based systems. In considering a sampling of existing agent-oriented architectures, we point to the need for a scalable and robust architecture capable of dealing with the many facets of agent-based systems. Accordingly, we specified a generic ring-based architecture that is intended to satisfy the practical applications.

We described “an agent-based curve-digitizing approach to well-logging data manipulation” as an example of a modeling and simulation environment for agent-based systems that was developed in accordance with the proposed architectural principles. While currently application does not offer the full spectrum of capabilities for an agent-based system, it provides sufficient evidence for the viability and applicability of the proposed architecture. Our implementation of “an agent-based curve-digitizing approach to well-logging data manipulation” is progressing toward the higher levels of agency needed to complete its capability to support modeling and simulation of systems in which agents play a major role. It will then offer a proof of concept for the architecture we present here.

The result of this research shows that the ring-based matchmaker architecture can be used for matching task agent and middleware agent; furthermore, the ring-based matchmaker architecture can be used as a basis for the agent-based systems that need to use facilitators. The main contributions in this paper are as follows.

- A scalable and robust framework that satisfies both agent-based multiple systems and heterogeneous database environment is built.
- The ring-based architecture model of agent-oriented matchmakers is developed and implemented.
- A practical application with the ring-based matchmaker architecture is developed.

References


