ATLANTIS PRESS International Journal of Networked and Distributed Computing, Vol. 6, No. 2 (April 2018) 88-98

Real-time Multi-Agents Architecture for E-commerce Servers

Sylvanus A. Ehikioya[1], Cong Zhang[2]

[1] *Department of Computer Science, Baze University,*
Abuja, NIGERIA
E-mail: ehikioya@gmail.com

[2] *Department of Elect & Comp. Engineering,*
University of Manitoba
Winnipeg, MB, Canada R3T 2N2
E-mail: czhang@ee.umanitoba.ca

Abstract

Electronic commerce applications have strict timing constraints on the interactions between e-commerce servers and customers. Customers prefer real-time services, which mean immediate response from the server shortly after submission of a request. This trend of demand gives e-commerce servers high pressure, both on the software and the architecture they currently use. In this paper, we propose a real-time architecture for e-commerce servers that addresses the problem efficiently. We adopt a framework that permits the appropriate treatment of dynamic behaviours that are data interdependent, and reasoning about the communication protocols and internal mechanisms of client / server relationships in a real-time multi-agents based e-commerce application architecture.

Keywords: Electronic commerce, Multi-Agents, Real-time, Transactions.

1. Introduction

With "electronic commerce" becoming a very common term to both customers and business partners, and governments of many countries, all kinds of electronic commerce (e-commerce for short) applications are booming on the Internet. Although these applications offer consumers enormous choices for product lines and comparison product pricing capabilities, amongst other numerous benefits, and provides e-commerce merchants great opportunities to showcase their products (goods and services) and tremendously increase their revenue base in the past few years, they are not quite robust. Both business-to-business (B2B) and business-to-consumers (B2C) applications have strict timing constraints on when and how long e-commerce application / server can finish a transaction, customers would not like to wait for a long time to place order for their goods on a merchant's e-commerce website. Thus, the timing constraint becomes a critical issue in current e-commerce applications.

E-commerce systems are reactive and cut across multiple geographical locations and platforms, and exhibit similar characteristics of some real-time systems and distributed systems. According to Douglas[1, 2], real-time systems' characteristics and issues include:

- Timeliness
- Concurrency
- Predictability
- Efficiency
- Distribution and communication
- Fault tolerance including reliability and safety
- Hardware interfacing

It should be noted that although e-commerce systems can be classified as real-time systems, they are generally, overall, soft real-time systems compared to hard real-time systems where the strict fulfillment of the timing constraints associated with the system's

functionality is critical in the decision to accept the outcome of a processing activity or not.

In a distributed system, components located on networked computers interact with one another (communicate and coordinate their actions by passing messages) in order to achieve a common goal. Some of the key features of a distributed system are resource sharing, openness, concurrency, scalability, fault tolerance, and transparency[3]. Additional features[4] include: quick response time, high throughput, high reliability, and easily expandable (or modular expansion). These are possible because of simultaneous execution of many processes at different computers (possibly at different locations).

E-commerce systems / applications must be responsive. A non-responsive e-commerce website is a turn-off for potential and current customers, thereby making the owner merchant loose customers, potential purchases (money), and market share. "But responsive Web design is not only about adjustable screen resolutions and automatically resizable images, but rather about a whole new way of thinking about design[5]". Although responsive web design is a very challenging feature for today's websites, it must offer service provisioning within time constraints, be robust, and fail safely, and allows websites to adapt in different viewport sizes nicely. Rams[6] provides the ten commandments of good design while The Interaction Design Foundation[7] lists seven factors (accessible, credible, desirable, finable, useful, usable, and valuable) that influence user experience. All these help provide great user experience in the use of digital products. Great user experiences translate into user pleasurable experiences that "create the least amount of friction while delivering fluid, seamless interaction and anticipator experiences; i.e. having things appear as if by magic. The right things, in the right moment, in the right way[8]". We abstract these responsive design issues in this paper but focus on the practicalities of the craft, the technique and processes involved in running a user-centred design project (whether the project is agile or waterfall) in order to maintain focus. Interested readers should see the following resources[5, 6, 7, 8, 9, 10, 11] for the details.

In this paper, we present the design of real-time multi-agents e-commerce servers' architecture, which includes the components of real-time multi-agents architecture (e.g., RTCustomerAgent, RTCardProcessingAgent, RTBankAgent, RTSearchingAgent, RTReportingAgent, RTWarehouseAgent, and RTShippingAgent) and how real-time communication is achieved by adding timing constraints / requirements. Since communication between agents is done in real-time, the efficiency of e-commerce transactions will be highly improved.

We adopt a framework that permits: (a). The appropriate treatment of dynamic behaviours that are data interdependent, and (b). Reasoning about the communication protocols and internal mechanisms of client / server relationships in a real-time multi-agents based e-commerce application architecture.

By using a script (language) processor to abstract communication protocols and internal mechanisms of client / server relationships in the real-time multi-agents architecture, it makes it possible to specify critical system components and behaviours, thus enabling formal reasoning about multi-agents. It makes it possible to reason about and prove both static and dynamic aspects of transactions, such as correctness, concurrency, safety and liveness, timing relationships as well as dependency relationships among transactions. The ability to control, monitor, and dynamically spawn agent coordinated events from a script adds to the usefulness, and helps to clarify formal action definitions within an agent network. The combined functionality of an agent definition script and that of the underlying preexisting programming language, such as Java, also adds structure to the formalisms on multi-agent transaction systems.

The contributions of this paper are: (i) Properly contextualizes e-commerce systems / applications key features with those of real-time systems and distributed systems; (ii) Provides an architectural model of e-commerce applications server with real-time features; and (iii) The implementation of our design provides a test-bed platform / environment for learning and gaining practical experience in real-time multi-agents architecture (software) design and implementation.

The rest of this paper is organized as follows: Section 2 examines the key features of e-commerce systems, which offers the critical link to real-time systems and distributed systems. Section 3 provides background contexts such as agent, multi-agent, agent communication languages, real-time systems and real-time constraints for proper understanding of our model. Section 4 describes our model of real-time multi-agent architecture for e-commerce servers. Finally, in Section 5 we discuss our future work.

2. Characteristics of Electronic Commerce Systems

E-commerce systems enable customers to make online purchases. A typical ordering process in business-to-

customer e-commerce systems allows customers to search and find items to purchase, negotiate the price of items, add items to a shopping cart, checkout items (i.e., purchase items), and pay for items purchased; the system also allows e-commerce merchants to update their inventory, verify customers' payment methods and plan logistics for shipping items to the customer.

E-commerce systems are inherently complex. According to Ehikioya[12], the complexity of e-commerce systems results from the concurrent, distributed, dynamic, and real-time behaviour and complex data access patterns of e-commerce transactions:

(i) *Concurrency*: Many processes (a unit of concurrent activity) in e-commerce transactions may execute concurrently. This interaction may involve communi-cation, synchronization, cooperation, parallelization, and competing for resources with other processes and the environment. For example, debiting a customer's credit card account and crediting the e-commerce service provider's credit card company could occur concurrently but transparent to both the user and merchant. However, concurrency control mechanisms (including serializations) must be in place to preserve transaction isolation.

(ii) *Distribution*: E-commerce systems are inherently distributed. An e-commerce application is usually distributed over possibly heterogeneous databases, web servers, networks, and operating systems across various locations. E-commerce transactions are characterized by complex data access patterns. Also, distributed e-commerce transaction processes can be invoked directly or indirectly from remote locations.

(iii) *Dynamism*: Dynamic systems have a number of states and changes are made to these states from time to time. When e-commerce transactions occur, various database tables are updated, therefore, changing the database state. In e-commerce systems, this dynamic behaviour results in data dependencies as well as the need for proper synchronization and communication among the various subcomponents of the system.

(iv) *Real-time Behaviour*: E-commerce transactions occur in real time. Real time implies the interaction between the e-commerce system and its environment occurs instantaneously. The real-time behaviour of e-commerce is influenced by the input environment and the application processing requirements. The system captures input from customers in real time. The real-time requirements of a system usually make some constraints on the input environment and output environment of the system.

(v) *Complex data access patterns*: E-commerce transactions inherently involve multiple data accesses across possibly multiple independent autonomous heterogeneous domains possibly across multiple distinct geographic locations. The propagation of access rights to data must be controlled while each autonomous domain enforces its security provisions. Besides this type of data access patterns complexity, an e-commerce transaction can be nested, thereby creating complex data serialization rules. A detailed examination of the data access patterns in e-commerce transactions along with correctness enforcement protocols is available in the following resources[12, 13].

Recall, e-commerce applications provide various back-end transactional processing and information access services across many heterogeneous databases and different networks which requires complex data access and interaction relationships in order to fulfill a discrete e-commerce objective. While the number of users in the business-to-business domain is much smaller than the business-to-consumer domain (e.g., the business-to-consumer domain may involve several million users whereas business-to-business may involve just a few hundred or thousand users), both require a highly reliable system that always guarantees consistent and correct results.

Additional essential attributes of e-commerce applications have been identified[14]. In particular, these attributes make e-commerce applications attractive candidates for formal modelling. However, these attributes will not be discussed in this paper.

3. Background Literature

We provide the following background material to provide context for the design in Section 4.

3.1. Agents

Agents-based framework and support for e-commerce transactions has gained enormous popularity and has become a common feature of commerce on the Internet. For example, many agents-based

implementations of many different aspects of e-commerce systems have been reported in the literature[15, 16, 17, 18, 19, 20].

An agent is an abstraction of software entity that acts for a user or other programs and it is "capable of acting with a certain degree of autonomy in order to accomplish tasks on behalf of its host[21]". In other words, an agent is a set of computers environment that is responsible for one or more specific kind of tasks. An agent is independent and autonomous, it can perform its function without the help of other agent or human beings, but it can communicate (and / or cooperate) with other agents to accomplish a task. The introduction of the "agent" concept makes it easy to control the design of the whole system.

Many definitions of the properties and functionality of an agent exist. Wooldridge and Jennings[22] describe agents as problem solving entities that have autonomy, social ability, pro-activeness and responsiveness. Autonomy is the ability of an agent to act on its own without any interference from users or other agents on its prescribed tasks. Social ability allows the agent to communicate with other agents or users. Pro-activeness refers to the ability of an agent to initiate some action when appropriate. Finally, responsiveness implies that agents understand their environment and can react accordingly to changes in it. Ehikioya and Walowetz[15] gave similar criteria for an agent in addition to agents' mobility characteristics, which means an agent can move throughout the network. Christoffel *et al.* argued[23] that agents allow users to interact in a uniform manner with a complicated system and help combat information overload while representing their interests. Similar properties of agents are available[19, 24, 25, 26] in the literature. According to Guttman and Maes[27] and Maes *et al.* [26], agents system can model the behaviours of its user through their intelligence. Intelligent agents are designed to perform analysis and make decisions based on the user's best interests[24]. The flexibility and enormous capabilities of agents make them suitable technology for e-commerce transactions negotiations[28].

We do not examine the features of agents in this paper beyond that necessary to model e-commerce transactions. We draw on available agents communication theory in the literature. It suffices, however, to note that multi-agents systems focus on cooperation and collaboration, joint goals and plans, and information sharing. This is synonymous to the phenomenon in the termite colony[15] model. The authors assume readers' familiarity with the agent paradigm.

3.2. Multi-Agent

A multi-agent system consists of multiple agents. Although agents are autonomous, in order to accomplish complex tasks, they need to communicate with one another and cooperate together. The advantage of using multi-agents is that it offers a bouquet of multiple solutions and multiple services. For every communication, there is one agent sending request (called requestAgent) and multiple agents serving the request (called serviceAgent). The requestAgent may broadcast its request with the service it is expecting, then serviceAgents receive the request, based on the parameter of the request, they check their available service, if their service meets the request, they respond to the requestAgent. The requestAgent may receive multiple replies from the serviceAgents; however, it chooses the most appropriate serviceAgent to perform its task.

Our multi-agents system architecture draws from the knowledge and theories of service oriented architecture (see Fig. 1a.) well established in the literature[29, 30], whereby services can be provided locally or outsourced to external service providers. Our architecture uses the simple object access protocol (SOAP) standard (a message exchange standard that supports service communication) and web service definition language (WSDL) standard that allows a service interface and its binding definitions. The binding maps the abstract interface to a set of protocols that specifies how (or rules) to communicate with a web service, a fundamental ingredient of web-based applications, including e-commerce systems.

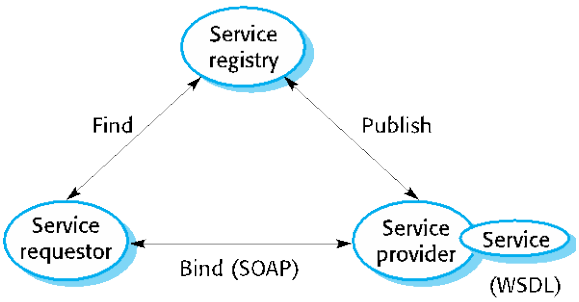

Fig. 1a. Service Oriented Architecture[29]

In most cases, this request-and-reply communication is not done directly between requestAgent and serviceAgent. A broker is introduced to deal with the multiple requests and multiple services. At the beginning, every serviceAgent registers its available services including name, OS, reliability, QoS, available time, etc. with a broker. When requestAgent has a task

with some parameters (intended OS, reliability, etc.) to be executed, it will not send this request directly to multiple serviceAgents, instead, it sends its request to the broker. The broker matches requestAgent's request with serviceAgents's registration information, if it finds matched pair, it will forward requestAgent's request to the appropriate serviceAgent available. Thereafter, the requestAgent could communicate with the selected serviceAgent directly. Fig. 1b. describes the communication procedure among multi-agents.

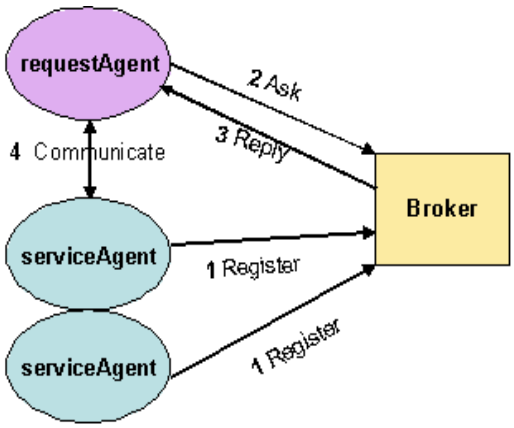

Fig. 1b. Communication among Multi-Agents

The introduction of the broker simplifies the communication between requestAgent and serviceAgent, and reduces communication time and overhead. For a detailed examination of multi-agents in e-commerce environments, interested readers should see the following resources[15, 28, 31].

3.3. Agent communication language

As we mentioned in Subsections 3.1 and 3.2, agents communicate with one another. Communication among multiple agents requires a framework and protocol that governs how agents interact with one another. The agent communication language provides the framework and protocol for interactions among agents. Communication between any two agents is akin to communication between two networks at the same network level, both ends should follow the same standard protocol.

3.3.1 Knowledge query and manipulation language

Knowledge Query and Manipulation Language (KQML) is a popular agent communication language and protocol for exchanging information and knowledge. KQML is both a message format and a message-handling protocol to support run-time knowledge sharing among agents[32]. KQML is a widely used agent communication language, and it provides "performatives" / primitives to define the interactions between agents. Examples of such primitives are:

Advertise:	service agents register services with broker agent
Ask	request agent requests service agents to perform a task
Tell:	provide other agents with its service information
Monitor:	watch another agent for a particular condition
…	

For example, a message representing a query about price of Air Canada tickets might be encoded[33] as:

(ask
:content(PRICE AirCanada?price)
:receiver TicketQueryBroker
:language LPROLOG
:ontology MYSE-TICKS)

3.3.2 Notation Agent Language (NAL)

In this paper, we focus on the application layer, thus we do not examine the details of how the agent communication language is implemented and linked to lower layers (middleware and operating system). To simplify our discourse, we introduce Notation Agent Language (NAL) to describe the communication between multiple agents.

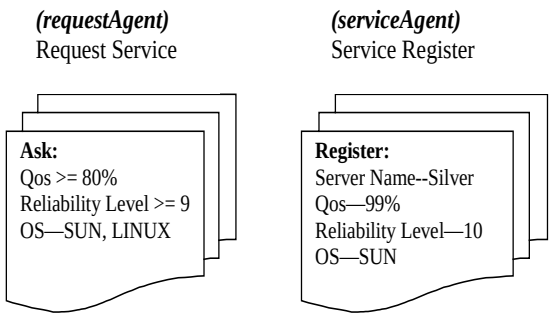

Fig. 2. Notation Language of Multi-Agents

The NAL is flexible and amenable to use for formal modeling of agents-based systems (see Fig. 2. above).

We extended its basic primitives to support real-time features to suite the modeling and specification of the real-time multi-agents systems. See Section 3.5 for a brief basic introduction.

3.4. Real-time systems

A real-time system is a system where the correct

functioning of the system depends on the results produced by the system and the time at which these results are produced[34]. Real-time systems are generally categorized into two types:

- *Soft real-time systems*, where operation is degraded if results are not produced according to specified timing requirements.
- *Hard real-time systems*, where operation is incorrect if results are not produced according to the timing constraint specification.

In this paper, we focus on soft real-time systems. When the server is unable to finish its tasks in the specified time period, the operation is degraded; that is, the QoS is degraded.

3.5. Real-time constraints

As discussed in Section 1, many e-commerce applications rely on real-time features, so they require service provisioning in real-time; otherwise the QoS will degrade significantly. To finish a task in real-time, the requestAgent should submit its request with specification such as "you should finish the job in 2 seconds" or "you should finish the job before 5:50pm March 26, 2002, CST", etc. These specifications are called timing constraints. By adding timing constraints to the request, the broker could find the appropriate serviceAgent to perform the job efficiently.

The timing constraints could be expressed in different languages. DiPippo *et al* [33] extended KQML performatives with expressions of time for both specification of timing capabilities and constraints. An example of such extension, following the style provided[33], is given below:

```
(ask
    :content(PRICE AirCanada?price)
   :receiver TicketQueryBroker
   :language LPROLOG
   :ontology MYSE-TICKS
    :deadline 3 seconds)

(register
    :language Prolog
    :content(PRICE ?x ?y)
    :exectime 2 seconds)
```

In a similar way, we could extend our NAL to RT-NAL by add timing constraints in the format shown in Fig. 3.

In RT-NAL multi-agents communication system, when *RTserviceAgent* registers its service with a broker, it includes "Execution Time -- 2 sec". When a *requestAgent* sends a request to broker, it includes "Deadline -- 3 sec".

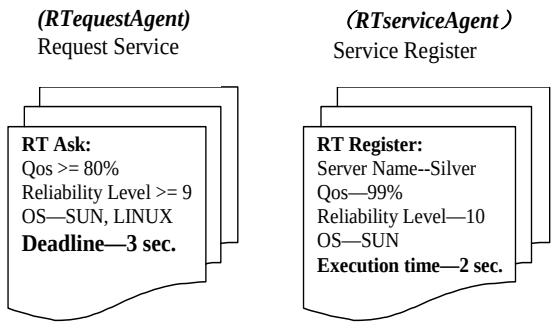

Fig. 3. Real-time Notation Language for Multi-Agents

The broker compares the two parameters of the *serviceAgent* and *requestAgent* to determine if the following condition is supported:

$$RTAsk.exectime <= RTRequester.deadline$$

That means the *serviceAgent* could perform the service asked by the *requestAgent*. In this way, the broker would find the best service for the requester and ensures the guarantee of real-time service provisioning.

4. Real-Time Architecture for E-Commerce Servers

In this section, we describe the methodology and design of our model of real-time multi-agent architecture for e-commerce servers.

4.1 Problems in the current architectures

We adopt a very simple working example to illustrate the problems in the current architectures; however, it is expressive enough to provide the solution directions and motivation for our design. Consider an example, a customer Nancy wants to buy an air ticket for an emergency meeting which will be held five days later in Taiwan:

1. She went to an airline online booking website, clicked the "search" button, after 60 seconds, she gets the search results (violates ***real-time searching*** policy);

2. She found a proper flight, she input her personal information and credit card details, and then clicked the "submit" button. There is no response whether her credit card is valid or not, whether she has enough money or not. After two days, she received an email that her card number is incorrect, so she

needs to submit the order form again (violates the ***real-time credit card processing*** policy);

3. Nancy had to go to the website again to fill in order form. This time, she is very careful, she typed all the characters and numbers one by one, then clicked the "submit" button very carefully. After 30 seconds, she receives a receipt from the website (violates the ***real-time reporting*** policy);

4. Then, Nancy kept waiting and waiting, waiting for the air-ticket she has ordered, this time, she did not receive any email, even no email ask her to type her credit card again, may be that means she was correct the last time. But with time flying, she became nervous, there is only one day left before the meeting. Finally, she received her ticket on the 6th day, but the meeting has been missed, what a pity. This is caused by the non-real-time shipping system, the air ticket warehouse did not receive customers request immediately (violates the ***real-time shipping*** policy).

Through this example, one could imagine the level of frustration and disappointment Nancy had, and thus criticality of real-time support for e-commerce servers.

4.2 The real-time architecture

To resolve the above embarrassing situation, we propose a real-time multi-agents architecture, illustrated in Fig. 4.

In this architecture, we introduce multiple real-time agents: *RTCardProcessingAgent*, *RTCustomerAgent*, *RTSearchingAgent*, *RTReportingAgent* and *RTShippingAgent*. Let us consider the same situations as in Subsection 4.1, but this time she goes to a real-time e-commerce website:

1. *RTCustomerAgent* is the representative of customers; it can communicate with other agents in real-time. Once Nancy logs in to a website, she is served by an RTCustomerAgent. When she searches for an air-ticket, the RTCustomerAgent sends her request to RTSearchingAgents with a timing constraint — "deadline 10 sec", the RTSearchingAgent with "execution time <=10 sec" contacts Warehouse in real-time and sends responses to RTCustomerAgent, so Nancy receives the search results within 10 sec. The non-real-time search problem is now addressed.

2. Nancy selects the proper flight, and completes the application form, and her credit card details, the RTCustomerAgent sends her information to the agent responsible for card processing, the RTCardProcessingAgent. The next step is that the RTCardProcessingAgent processes her card immediately by using a Real-time Credit Card Processing Architecture (see Fig. 5.). After 5 seconds, she receives a notice "Your Card is not valid, please check your card number again…" She makes mistakes again, but this time she is notified in real time. So she can re-fill the order form immediately, she does not have to wait for two days. Thus, the non-real-time credit card processing problem is addressed.

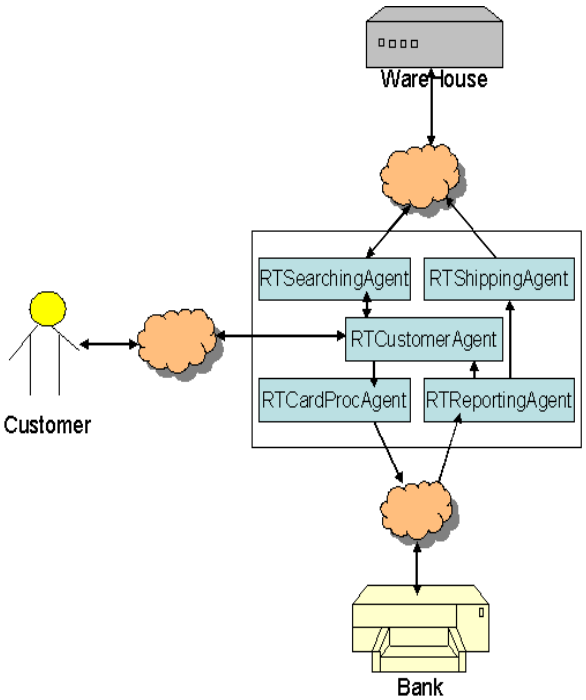

Fig. 4. Real-time Multi-Agents Architecture for E-commerce Servers

3. After Nancy refill her card information, RTCardProcessingAgent checks her card with Bank in real time, within in 5 seconds, Nancy is notified that her card is verified, she could proceed with her shopping.

4. Nancy's order form is submitted to the RTShippingAgent with the timing constraint "response in 10 sec, ship in 1 day". RTShippingAgent receives the order form, it does real-time response, Nancy receives the order receipt within 10 senconds, and receives her tickets in one day. Thus, the non-real-time reporting and non-real-time shipping problems are all well addressed.

4.3 Class pseudo code of the real-time architecture

We provide the pseudo-code of the classes in our architecture.

4.3.1 Real-time searching class

The RTCustomer Agent is a requestAgent, while the RTSeachingAgents are serviceAgents.

```
Public Class RTSearching {
    // RTSearchingAgents register their service abilities to RTBroker
    File RTSearchingAgent.register(QoS,
                Reliability, OS, ExecuteTime)
    sendFile(serviceFile, BrokerIP);

    // RTCustomerAgent is activated by a customer, then it asks available service from RTSearchingAgents
    RTCustomerAgent.activate();
    File  RTCustomerAgent.ask(QoS, Reliability,
                OS, TimeConstraints);
    sendFile(requestFile, BrokerIP);
    … …
    //RTBroker compares the requestFile and serviceFile to find the proper service agent
    String RTBroker.compareTo(requestFile,
            serviceFile)
    {return proper RTSearchingAgent machine's
                name};
    //RTCustomerAgent connects to the proper RTSearching Agent
    RTCustomerAgent.connectTo(RTSearching-
            Agent);
    //the selected RTSearchingAgent searches warehouse with specific keywords, return the available stocks
    Vector RTSearchingAgent.search(Warehouse,
            keywords)
    {return warehouse stocks};
}
```

4.3.2 Real-time credit card processing class

The real-time credit card processing is more complex than other real-time aspects of e-commerce applications. It not only involves adding timing constraints to the communication, but it also involves a new architecture design to address real-time credit card processing problem.

Real-Time Credit Card Processing means that when a web site's customer conducts an online purchase, the credit card information is conveyed to the Processor at that exact time so that an authorization can be requested and received at that moment. Real-time processing always implies that a Secure Payment Gateway is being utilized, whether proprietary or third party[35]. In our real-time credit card processing architecture design (see Fig. 5.), a third party is introduced to do real-time card verification and processing.

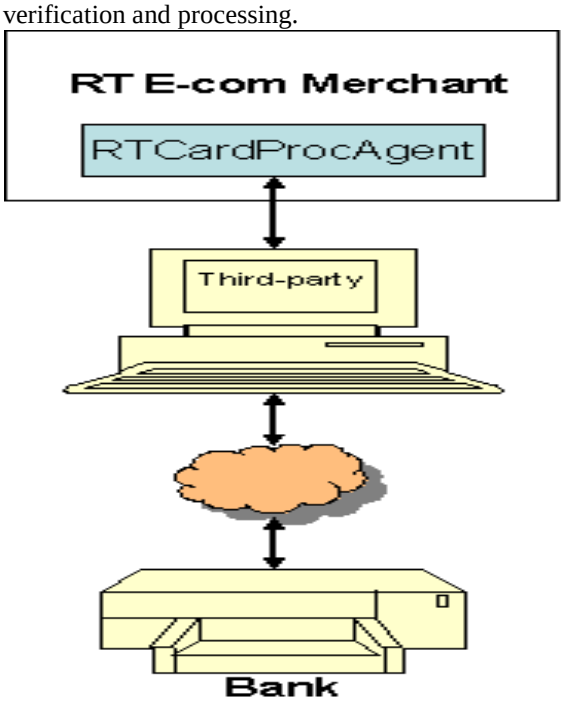

Fig. 5. Real-time Card Processing

When the RTCardProcAgent receives a card processing request, it forwards the request to the third party, the third party can process the request and verify the request immediately, thus, in 5-10 seconds, the receipt will be shown on customer's web page or sent to customer's email.

In this way, the following actions will be performed in real-time and parallel but as an atomic unit:

Credit(MerchantAccount, amountOfMoney);
Debit(CustomerAccount, amountOfMoney);
InformShippingAgent(Warehouse)

Real-time Credit Card Processing benefits both customers and merchants. For customers, they can finish their shopping in a short period, and receive their purchases very quickly. For merchants, they can sell their goods quickly and get back revenues.

The pseudo codes are as follows:

```
Public Class RTCardProcessing {
    // RTCardProcAgents register their service abilities to Broker
    File RTCardProcAgent.register(QoS,
```

```
                    Reliability, OS, ExecuteTime)
sendFile(serviceFile, BrokerIP);

// RTCustomerAgent is activated by a customer, then it ask
available service from RTCardProcAgents
RTCustomerAgent.activate();
File RTCustomerAgent.ask(QoS, Reliability,
                    OS, TimeConstraints);
sendFile(requestFile, BrokerIP);
… …
//RTBroker compares the requestFile and serviceFile to
find the proper service agent
String RTBroker.compareTo(requestFile,
                    serviceFile)
{return proper RTCardProcAgent machine's
                    name};

//RTCustomerAgent connects to the proper RTCardProc
Agent
RTCustomerAgent.connectTo(RTCardProc-
                    Agent);

//the selected RTCArdProcAgent forward
 RTCustomerAgent's request to the third party.
RTCardProcAgent.forward(thirdPartyIP,
                    cardInfo, AmountOfMoney)

// Thrid Party validate the credit card information
String ThridParty.Validate(cardInfo,
          AmountOfMoney, RTCardProcAgentIP)
{return the validation result;}

//Third party process the crediting the merchant account
and debiting customer account
ThridParty.process(AmountOfMoney,    MerchantAccount,
CustomerAccount){
    Credit(MerchantAccount, amountOfMoney);
    Debit(CustomerAccount, amountOfMoney);
  };

//Third party inform RTCardProcAgent and
 RTReportingAgent the amount of money charged
ThirdParty.inform(RTCardProcAgentIP, amountOfMoney)
}
```

3.3.3 Real-time reporting class

```
Public Class RTReporting {
  // RTReportingAgents register their service abilities to
   Broker
  File RTReportingAgent.register(QoS,
                    Reliability, OS, ExecuteTime)
  sendFile(serviceFile, BrokerIP);

  // RTCardProcrAgent  ask available service from
   RTReportingAgents
  File RTCardProcAgent.ask(QoS, Reliability,
                    OS, TimeConstraints);
  sendFile(requestFile, BrokerIP);
  … …
  //RTBroker compares the requestFile and serviceFile to
   find the proper service agent
  String RTBroker.compareTo(requestFile,
                    serviceFile)
  {return proper RTReportingAgent machine's
                    name};

  //RTCardProcAgent connects to the proper RTReporting
  Agent
     RTCardProcAgent.connectTo(RTReporting-Agent);

  //the selected RTCardProcAgent inform
   RTReportingAgent's with the amount of money charged
  RTCardProcAgent.inform(RTReportingAgent
                    IP, amountOfMoney)

  //Generate report and display
  Vector RTReportingAgent.generateReport();
  Display on the screen;
}
```

4.3.4 Real-time shipping class

```
Public Class RTShipping {
  // RTShippingAgents register their service abilities to
   Broker
  File RTShippingAgent.register(QoS,
                    Reliability, OS, ExecuteTime)
  sendFile(serviceFile, BrokerIP);

  // RTReportingAgent  ask available service from
   RTShippingAgents
  File RTReportingAgent.ask(QoS, Reliability,
                    OS, TimeConstraints);
  sendFile(requestFile, BrokerIP);
  … …
  //RTBroker compares the requestFile and serviceFile to
   find the proper service agent
  String RTBroker.compareTo(requestFile,
                    serviceFile)
  {return proper RTShippingAgent machine's name};

  //RTReportingAgent connects to the proper
   RTShippingAgent
  RTReportingAgent.connectTo(RTShipping-Agent);

  //the proper RTReportingAgent inform RTShippingAgent
   with the order information
  RT.ShippingAgent.inform(warehouse, order)
```

```
    //the RTShippingAgent inform warehouse to ship the goods
     to customer
    RT.ShippingAgent.inform(warehouse, order)
    Warehouse.decreaseGoodsAmount(numberOfOrder)
    Warehouse.shipToCustomer();
}
```

We use a script processor (compliant with the SOAP standard) to abstract communication protocols and internal mechanisms of client / server relationships in the real-time multi-agent architecture. Ehikioya and Walowetz[15] formally defined a script language, Multi-Agent Processing Language (MAPL), for reasoning about multi-agents. A scripting language provides a method of accomplishing multi-agents interdependence. Having the ability to control, monitor, and dynamically spawn agent coordinated events from a script adds to the usefulness, and helps to clarify formal action definitions within an agent network. The combined functionality of an agent definition script and that of the underlying preexisting programming language, such as Java, also adds structure to the formalisms on multi-agent transaction systems.

We note that the multi-agents in our system being mobile, they can travel from one site to another in a network performing tasks on behalf of the service request; e.g., collecting information, accessing database resources, and returning the result. Furthermore, they may make many invocations to local resources at each site they visit while executing an e-commerce transaction.

5. Summary and Future Work

The rising popularity of e-commerce will continue unabated in the foreseeable future and consumers increasing demand for better quality of experience from service providers, including e-commerce merchants will remain high. Thus, the QoS provisioning in most current e-commerce applications is inadequate and there is need for real-time support by e-commerce servers, as a matter of conscious design. We have demonstrated this as practicable and crucial in meeting certain e-commerce transactions' requirements in today's everyday sophisticated consumers' transactions requests.

Although the real-time e-commerce servers architecture has been proposed and partially implemented, more detailed implementation should be pursued in future by combining some middleware and operating system support to realize some of the critical real-time features.

References

1. B. P. Douglass, Real-Time UML, in *Formal Techniques in Real-Time and Fault-Tolerant Systems*. Lecture Notes in Computer Science, Vol. 2469, eds. W. Damm and E. R. Olderog (Springer, Berlin, Heidelberg, 2002).

2. B. P. Douglass, *Real Time UML: Advances in the UML for Real-time Systems*, 3rd Edition (Addison-Wesley Professional, 2004).

3. G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5th Edition (Addison Wesley / Pearson Education, May 2012).

4. Er. R. Chopra, *Operating Systems: A Practical Approach*, Revised Edition (S. Chand and Company Ltd, New Delhi, India, 2012).

5. Smashing Editorial, "Responsive Web Design: What It Is And How To Use It", *Smashing Magazine*, January 12th, 2011. (Available at: https://www.smashingmagazine .com/2011/01/guidelines-for-responsive-web-design/). Accessed on February 23, 2017.

6. D. Rams, Ten Principles for Good Design, (Available at http://www.manifestoproject.it/ten-principles-for-good-design/) Accessed on June 29, 2017.

7. *The Basics of User Experience Design*, The Interaction Design Foundation, (available at http://www.interaction-design.org). Accessed on June 29, 2017.

8. M. Philips, *Never Just Design Pretty Little Apps*, (Principal User Experience Designer, March 21, 2017). (Available for free download at *https://blog.prototypr.io /@miklosphilips*) Accessed on June 29, 2017.

9. J. Moule, *Killer UX Design* (SitePoint Pty Ltd, 2012).

10. T. Firdaus, *Responsive Web Design by Example: Beginner's Guide* (Packt Publishing, 2017).

11. J. Lang and E. Howell, *Researching UX: User Research* (SitePoint Pty Ltd, 2017).

12. S. A. Ehikioya, *Specification of Transaction Systems Protocols* (Ph.D. Thesis, University of Manitoba, Winnipeg, 1997).

13. S. A. Ehikioya and K. E. Barker, A Formal Specification Strategy for Electronic Commerce, in *Proceedings of IDEAS '97: International Database Engineering and Applications Symposium*, August 25-27, 1997 (Montreal, Canada, 1997).

14. S. A. Ehikioya and K. Hiebert, A Formal Model of Electronic Commerce, in *First International Conference on Software Engineering, Networking, and Parallel and Distributed Computing*, May 19-21 2000 (Champagne-Ardenne, France, 2000).

15. S. A. Ehikioya and T. Walowetz, A Formal Specification of Transaction Systems in Distributed Multi-Agents Systems. in *ISCA 14th International Conference on Computers and Their Applications*, (Cancun, Mexico, April 7-9, 1999), pp. 378-383.

16. AuctionBot. (Available: http://auction.eecs.umich.edu/)

17. M. Balabanovic, *Learning to Surf: Multiagent Systems for Adaptive Web Page Recommendation* (Ph.D Thesis, Department of Computer Science, Stanford University, Stanford, CA 94305-90250, March 1998).

18. A. Moukas and G. Zacharia, Evolving Multiagent Filtering Solution in Amalthaea, in *Proc. of the 1st International Conference on Autonomous Agents* (February 1997), pp. 394-403.

19. B. Aoun, Agent Technology in Electronic Commerce and Information Retrieval on the Internet, in *Proceedings of AUSWEB96*, 1996. (Also available at: http://www.scu.edu.au/sponsored/ausweb/ausweb96/tech/aoun/paper.html)

20. R. lnder, M. Hurst, and T. Kato, A Prototype Agent to Assist Shoppers, in *Proc. of WWW7*, 1997. (Available at: http://www7.scu.edu.au/programme/posters/1856/com18S6.htm)

21. Wikipedia, "Software Agent", (Available at *https://en.wikipedia.org/wiki/Software_agent*) Accessed Oct. 5, 2017.

22. M. Wooldridge and N. R. Jennings, Intelligent Agents: Theory and Practice, *Knowledge Engineering Review*, **10 (2),** pp. 118 – 152

23. M. Christoffel, S. Pulkowski, B. Schmitt. and P. Lockemann, Electronic Market: The Roadmap for University Libraries and Members to Survive in the Information Jungle, *SIGMOD Record*, Volume 27, #4, December 1998 (Special Section on Electronic Commerce).

24. J. Domingo-Ferrer and J. Herrera-Joancomarti, An Anonymous Electronic Commerce Scheme with an Off-Line Authority and Untrusted Agents, *SIGMOD Record*. Volume 27, #4, December 1998 (Special Section on Electronic Commerce).