ISSN: 2141 – 3290
www.wojast.com

PROTOTYPE OF AN INTELLIGENT TRADE AGENT

IMIANVAN A. A.[1] ; AKINYOKUN O.C.[2]
OBASOHAN E.E.[3] AND *OBI J.C.[4]
[1, 3 and 4]*Department of Computer Science,*
University of Benin, Benin City, Nigeria.
Federal University of Technology Akure. Nigeria.[2]
tonyvanni@yahoo.com[1] , *akinwole2003@yahoo.co.uk*[2] *and*
tripplejo2k2@yahoo.com[4]
**Correspondence Author.*

ABSTRACT: Intelligent software agents carry out tasks for users autonomously without direct intervention by its master once the tasks have been delegated. They are able to set plans and goals, reason about the effect of actions and improve their knowledge and performance through learning. With the use of intelligent agent technology, shoppers can attain faster and easier shopping on the Internet. In addition, they could be assured of a more thorough search to get desired products at a desired price. This paper has attempted to give a formal description of the activities an Intelligent Trade Agent. Using Zed notations an attempt is made to develop a trade agent that can shop on behalf of its user. The interaction within the system is visualized using Unified Modeling Language (UML) sequence diagrams whereas the autonomy of the agent is assured with Telescript command infrastructure.

INTRODUCTION

Software agent could be defined as "software using techniques from Artificial Intelligence (AI) to assist a human user of a specific application" (Franz, 2011). Intelligent software agents act on behalf of the user to find information, negotiate for services easily, automate complex tasks or collaborate with other software agents to solve complex problems (Dawn, 2011, Guy, et al 1999). Intelligent task centered agents can provide intelligent assistance to end-users and be a boom to productivity in the world of e-commerce (Olga, 2003).

More than a decade, there has been buying and selling over the Internet. However, formal specification of agent prototypes for trading over the Internet appears to be few. Consequently, this research attempts to formalize using Z (Zed) notations the specification of an intelligent agent to be deployed for e-trading. Some graphical facilities of the Unified Modeling Language were will be employed to visualize the interactions of the agent. Telescript command infrastructure guarantees the autonomy of the agent system.

The proposed system should be able to do the following:

a. The buyer agent will receive input from user in the form of 'item' and maximum price.
b. The buyer agent will also be able to buy items on behalf of the user when it finds the requested item at the maximum price specified by the user or at a lower price.
c. The seller agent will be able to update its catalogue with the items available at the entire store and the associated prices.
d. The seller agent will also be able to show the buyer a list of the available items.

FORMAL SPECIFICATION OF THE INTELLIGENT TRADE AGENT

Specification is the expression in some formal terms and at some level of abstraction, of a collection of properties, some system should satisfy. The historical perspectives and the qualities of good specifications have been discussed (Spivey, 1998). Z–Notation is a formal specification construct based on set algebra and predicate calculus for specification of computing systems (Spivey, 1992). It has the ability of decomposing specification into small piece called schemas. By splitting the specification into schemas, we can present it piece by piece.

The following are some of the basic types in Z.
[CHAR, STRING, CURRENCY, QUERY, OBJECTS, COMPONENTS, TIME, DATA and BOOLEAN::= TRUE/FALSE]. There are a number of currencies supported by the agents. Payments and receipt are made in these currencies. CURRENCY ::= POUND/USD/NAIRA.
Every user is authenticated using his user name and ID.

User
User_name: seq CHAR
User_ID: seq CHAR
User: ℙ USER
Access! : Boolean
(user ∈ user.access! = accepted) V (user ∉ user.access! = denied)

Figure 1: User Schema

There is no limit to the number of registered users and each user can have one or more agents acting on his behalf. Logging on, each agent must register its name and the service it offers with the AgentDirectory to enable other agents find it.

AgentDirectory
Agents: ℙ AGENT ; Services: ℙ SERVICE
Directory: AGENT → SERVICE
Agents = dom directory

Figure 2: AgentDirectory Schema

The directory lists all agents and services they render and enables other agents looking for a particular services to find the agent.

RegisterAgent
Δ AgentDirectory
agent? : AGENT
service? : SERVICE
report! : REPORT
(agent? ∉ agent ˄
AgentDirectory′ = AgentDirectory U {agent? → service?}˄ report! = ok) ˅
(agent? ∉ agent ˄ AgentDirectory′ = AgentDirectory ˄ Report! = already_known)

Figure 3: RegisterAgent Schema

The directory supports the facility to register an agent provided that agent does not already exist. If the agent exists, a report of 'already known' is returned. Agents can be found in the directory based on the type of service they render.

FindService
Ξ AgentDirectory
Service? : SERVICE
Agentlist? ℙ AGENT
(agentlist! = {agent : agents / directory (agent) = service? }˅ (agent ∉ agents ˄ report! = not_known)

Figure 4: FindService Schema.

The Findservice function receives a service type as an argument and returns all the agents who offer the service in a service list.

AlreadyExistingAgent

Ξ AgentDirectory
agent? : AGENT
report! : REPORT

agent? $\in$ Agents
report! = already_known

Figure 5: AlreadyExistingAgent Schema

ServiceNotAvailable

Ξ AgentDirectory
agents: $\mathbb{P}$ AGENT
report! : REPORT
service? : SERVICE

$\forall$ agent $\in$ agents**.**directory(agent) $\neq$ service
report! = not_known

Figure 6: ServiceNotAvailable Schema

The schema in Figure 6 shows that error which occurs when an agent requests for an agent list for a service, which has not been registered for, by any agent. An error report of 'not known' is supplied to the agent. The directory is initialized at the beginning with no agents and no services in Figure 7.

Initialize_Directory

AgentDirectory
Services: $\mathbb{P}$ SERVICE
agents $\mathbb{P}$ AGENT

agent = $\varnothing$
service = $\varnothing$

Figure 7: Initialize_Directory Schema

For online shopping purpose, each agent carries a shopping list of items to be purchased and the maximum price at which to purchase each item.

ShoppingList

Items: ITEM
Maxprice : $\mathbb{P}$ Price
Shoppinglist: ITEM $\rightarrow$ PRICE

Item dom (shoppinglist)

Figure 8: ShoppingList Schema

The function shoppinglist in Figure 8, links each item in the list to the maximum price, at which it may be purchased.

AddItem

Δ ShoppingList
Item? : ITEM
Maxprice? : PRICE
Report! : REPORT

(item $\notin$ items $\wedge$ ShoppingList$'$ = ShoppingList U { item $\rightarrow$ maxprice} $\vee$ Report! = known)

Figure 9: AddItem Schema.

Items can be added to the shopping list with their corresponding maximum purchase price.

```
──────────── RemoveItem ──────────
│ Δ ShoppingList
│   Item? : ITEM
│   Report! : REPORT
├──────────────────────
│ (item ∉ items ˄ ShoppingList′ = ShoppingList / {item → maxprice} ˅
│          (item? ∈ items ˄ report! = known )
└──────────────────────────────────
```

Figure 10: RemoveItem Schema

Note that a shopping list could be deleted that is, all the items in the list removed.
Agents have a number of message types they can send in the process of interaction with each other.

MSG ::= REQUEST / INFORM / QUERY_IF / CFP / PROPOSE /
ACCEPT_PROPOSE / REJECT_PROPOSAL

The REQUEST message asks the receiving agent to perform a certain action on behalf of the agent sending the message. The action could be to check the catalogue for a particular item and to return its price if it exists.
INFORM informs the receiving agent.
INFORM: STRING
QUERY_IF checks if a condition holds or not and returns TRUE if it does and FALSE, otherwise.
PROPOSE gives a proposal if the price of the item is below the maximum price limit for the item, otherwise, a REJECT_PROPOSAL message is sent.

A message is encapsulated in a message schema (Figure 11) containing the addresses of the agents receiving and sending the message, the data, the time, a message ID, the language, the currency and the message type.

```
┌──────── Message ──────────
│ From, To: AGENT
│ Date: DATE
│ Message: STRING
│ Msg? : MSG
│ Language: LANGUAGE
│ Currency: CURRENCY
│ Msg ID: ℤ
├──────────────────
│ 4 ≤ #msg ID ≤ ID
└──────────────────────────
```

Figure 11: Message Schema

The mailbox defines the part of an agent responsible for receiving and sending messages.
Messages are queued and retrieved for processing in the mailbox (Figure 12).

```
┌────── Mailbox ──────────
│ In, Out: ℙ message
│ queue: ℙ message
├──────────────────
│ in = queue
└──────────────────────────
```

Figure 12: Mailbox Schema

SendMessage

Δ Mailbox

out? = message

in′ = in ; out′ = out $\wedge$ out?

Figure 13: SendMessage Schema

'In' refers to the inbox of the agent which receive and stores messages, while 'out', refers to the outbox.

ReceiveMessage

Δ mailbox

in? : message

In′ = in $\wedge$ in? ; #in = #in + 1

Figure 14: ReceiveMessage Schema

The seller agent carries a catalogue of items for sale and their prices. The schema, AgentCatalogue (Figure 15), represents the agent's catalogue.

AgentCatalogue

Items: $\mathbb{P}$ ITEM

Prices: $\mathbb{P}$ PRICES

Catalogue: ITEM $\rightarrow$ PRICES

Items = dom catalogue

Figure 15: AgentCatalogue Schema

Items are added to the catalogue by the seller and, are removed when they have been sold (Figure 16).

AddItemToCatalogue

Δ AgentCatalogue

Item? : ITEM

Price: PRICE

(Item $\notin$ items $\wedge$ AgentCatalogue′ = AgentCatalogue $\wedge$ Report! $\neq$ already_known)

Figure 16: AddItemToCatalogue Schema

RemoveItemFromCatalogue

Δ AgentCatalogue

Item? : ITEM

(Item $\in$ items $\wedge$ AgentCatalogue′ = AgentCatalogue $\wedge$ Report! = already_known)

Figure 17: RemoveItemFromCatalogue Schema

SYSTEM DESIGN AND UNIFIED MODELING LANGUAGE (UML)

Software design is the translation of the requirement specification into useful patterns for implementation. Unified Modeling Language (UML) is a standard modeling language used for modeling software systems. We used UML for designing the trade agent because UML focuses on creating simple, well documented and easy to understand software models. UML sequence diagram shows the interaction between classes (or object) in the system for each use case. The interaction represents the order of messages that are exchanged between classes to accomplish a purpose. For the trade agent that has been specified (Fig. 18).

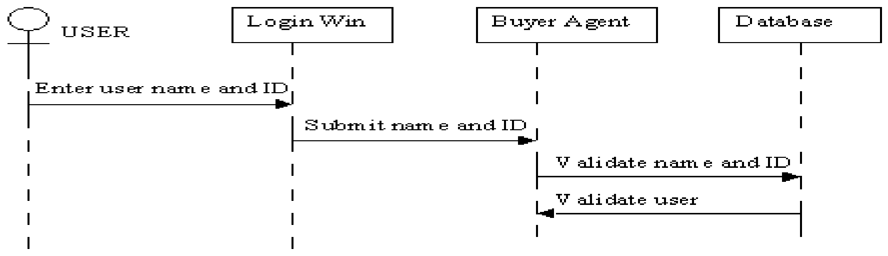


Figure 18: Login user sequence diagram

Figure 18, models the sequence of steps involved in the log in user case scenario. The order of appearance of the arrows indicate the order of the actions while the arrow direction indicates the direction of flow of events / results.

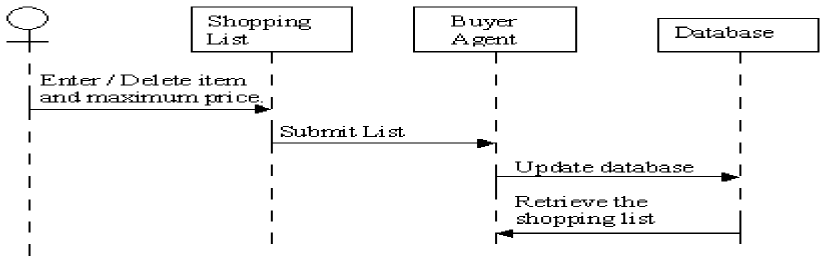

Shopping List
Buyer Agent
Database
Enter / Delete item and maximum price
Submit List
Update database
Retrieve the shopping list

Figure 19: Add / Remove Item sequence diagram.

In Figure 19, we specify how to add or remove any item from buyer agents list. Any changes made will be updated by the system and is saved into the agents database.

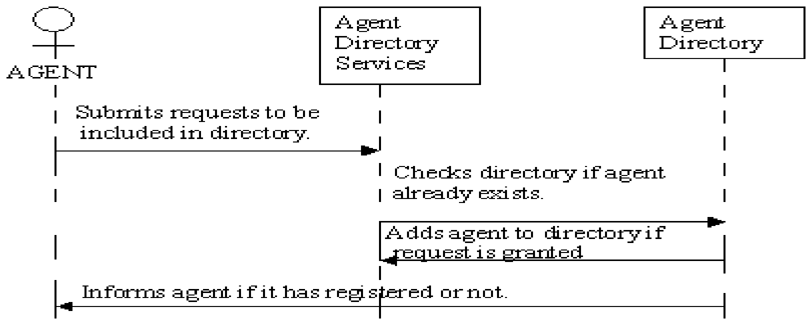

AGENT
Agent Directory Services
Agent Directory
Submits requests to be included in directory.
Checks directory if agent already exists.
Adds agent to directory if request is granted
Informs agent if it has registered or not.

Figure 20: Register Agent Sequence diagram.

Figure 20, specifies how to register an agent. The agent directory service checks if agents already exist in directory. If it does, registration is not allowed.

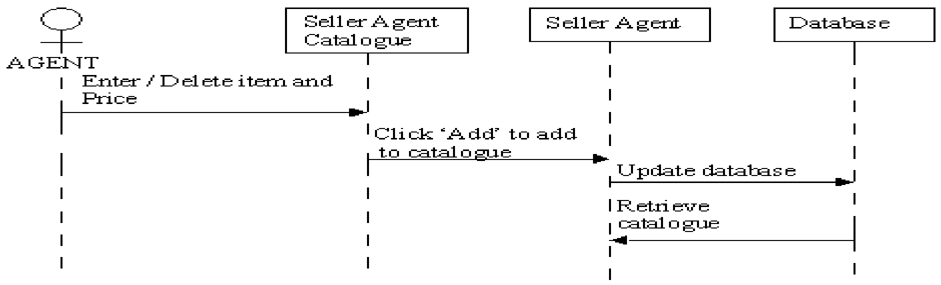
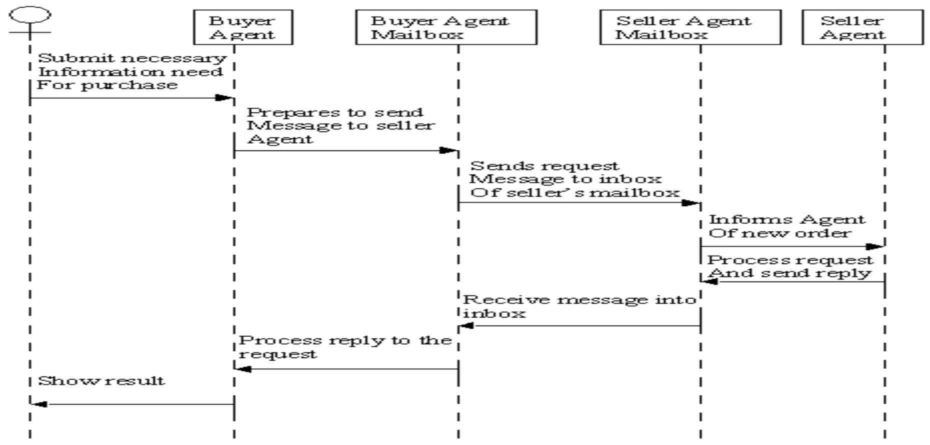


Figure 21: Add / Delete item for seller Agent sequence Diagram.

Figure 21, adds or removes item from seller agent catalogue and updates its database to show in catalogue, only available items and associated prices.



Buyer Agent
Buyer Agent Mailbox
Seller Agent Mailbox
Seller Agent
Submit necessary Information need For purchase
Prepares to send Message to seller Agent
Sends request Message to inbox Of seller's mailbox
Informs Agent Of new order
Process request And send reply
Receive message into inbox
Process reply to the request
Show result

Figure 22: Transaction Sequence Diagram.

Figure 22, shows how the buyer agent and seller agent communicate messages back and front through their mailboxes. The result is a transaction, where the user either gets the item requested at a favourable price or does not get the item at all.

Implementation

Using Telescript, the agent is captured as process that controls its movement from one *Place* process to another within the World Wide Web (WWW). The place is presented as a process that provides a computing context within which *Agent* processes are run. The agent and places were captured as subclasses of the Process class. Telescript Permit object is used to grant capabilities, such as the ability to create or clone agents to the trade agent system. Telescript Go and send command is used to transport the trade agent to the destination specified by the Ticket object. Telescript Self command is promoting autonomy of the agent. The trade agent is able to create copies of it self. Telescript meet command is allowing the agent to interact with other agents at a *Meeting Place.* The agents only interacts through a meeting (exchange references to each other). For example to request a meeting with an agent specified in a Petition object, we may write:

```
agentToMeet := here@MeetingPlace.meet(aPetition,nil);
```

The system is equipped with Graphical User Interface (GUIs) for both buyer agent and seller agent. Now we can update the catalogue in the seller agent by adding new available stock to it, that is, items and their prices, which is displayed and then clicking “Add” to add a product. The seller agents continuously wait for requests from buyer agents. When asked to provide an offer for an item they check if the requested item is in their catalogue and in this case reply with the price, otherwise, they refuse. When they receive a purchase order, they serve it and remove requested item from their catalogue.

On the buyer agent GUI we specify the list of items we want to buy and how much we are willing to spend for each of these items, method of payment and other necessary information needed for the transaction to kick off. As soon as an offer is received, the buyer agent accepts it and issues a purchase order. If more than one seller agent provides an offer, the buyer agent accepts the best one (lowest price). Having bought an item, the buyer terminates on going search for that particular item on the list and a new search can be kicked off for another item being added to the list.

The authors are on a detailed implementation of the trade agent system. Simulation results obtained from the Case Study of the intelligent trade agent proposed in this work will for the basis of a future paper.

FINDINGS

Based on the ease at which the users get information through this new system, the following are revealed:

a. The intelligent trade agent is able to trade on behalf of the user.

b. The intelligent trade agent system when implemented will save users time, as other jobs can be done by the user while the agent searches the web. Ordinary, the user would have had to do a manual search.

c. A user who is not very good at searching the web, can saddle the agent with that task. A disadvantage of using this agent, is the fact that some sites are designed to keep agents out and so, the buyer agent is not assured of actually getting the "best" bargain available.

CONCLUSION

Although, it is still a fairly new technology, using object oriented intelligent agent in e-trading, is neither science fiction nor should it be off putting. The impact of good intelligent agent systems on our world is profound. Static information systems fetch information only as ordered. New articles always match the same profile as the last ones did. In contrast, information systems with intelligent agents are dynamic. Intelligent trade agents can compare prices and chose the best price according to the user's criteria. They detect changes in the user's interests and alter to reflect those now facets. They can also monitor prices and products overtime. They will also detect new orders and offer to add it to the query. The query can be refined still further. The effect is that of a shopper spending less time shopping on the World Wide Web. The agent activities takes place at the background while the shopper does more important tasks like reading what has been retrieved, interpreting it, and acting upon result. Trade agents can be embedded in a commercial site to handle the users personal information as a secure transaction by a trusted third party.

REFERENCES

Dawn G. G., (2001): DSS Access on the WWW: An Intelligent Agent Prototype, http://aisel.isword.org/pdf

Franz K, (2011): Intelligent Agents-Cal Poly (ppt) retrieved from users.csc.calpoly.edu/-fkurfess/480/fu/slides/2-agent

Guy G., Wong J. S., Honavar V., and Miller L. (1999): Intelligent Agents for Intrusion Detection, http://dependability.cs.virginia.edu/bibliography/helmer98intelligent.pdf

Olga S., (2003): "Exploring Trading Dynamics in a Derivative Securities Market of Heterogeneous Agents", University of Maryland, Baltimore County.

Spivey J. M. (1992): The Z Notation: A Reference Manual, 2nd Edition, Prentice Hall International (UK) limited, United Kingdom.

Spivey J. M., (1998): The Z Notation: A Reference Manual, Oxford, J. M. Spivey.