ISSN: 2141 – 3290

FORMAL CHARACTERIZATION OF AN INTELLIGENT ACADEMIC RESPONSIBILITY ASSIGNMENT AGENT

OBI J.C[1]. AND IMIANVAN A.A[2].
[1&2] Department of Computer Science,
University of Benin, Benin City. Nigeria.
Email: tripplejo2k2@yahoo.com[1] and tonyvanni@yahoo.com[2]
+234(0)8093088218[1] and +234(0)7069742552[2]

ABSTRACT: Intelligent Software Agent perform autonomous and unanimous task for users based on predefined rules. They are able to attain goals, meditate on the consequence of an action and tune-up knowledge and performance through learning algorithm. Utilizing intelligent software agent, Head of Department can allocate departmental responsibilities with ease based on lecturer's abilities, specific qualification and experiences without undue altercation which is the crucial role of this paper. Using Zed notations, an attempt is made to develop software agents that can allocate departmental responsibilities on behalf of the Head of Department. The agent functions as e-education agent. The proposed system reduces problems often associated with responsibility assignment.

INTRODUCTION

Software Agent (or Autonomous Agent) is a computer program which works toward goals (as opposed to discrete tasks) in a dynamic environment on behalf of another entity, human or computational, possibly over an extended period of time, without continuous direct supervision, and exhibits a significant degree of flexibility and creativity in how it seeks to transform goals into action tasks (John, 2005) . It could also be defined as "software using techniques from Artificial Intelligence (AI) to assist a human user of a specific application" (Bjorn, 1996 and Maes, 1994). Intelligent software agent act on behalf of the user to find, filter information, negotiate for services easily, automate complex tasks or collaborate with other software agents to solve complex problem (Spivey, 1998; Stuart and Norvig 1995).

Academic freedom and responsibility have long been topics for public concern and debate. Academic freedom to explore significant and controversial questions is an essential precondition to fulfill the academy's mission of educating students and advancing knowledge. Academic responsibility requires professors to submit their knowledge and claims to rigorous and public review by peers who are experts in the subject matter under consideration; to ground their arguments in the best available evidence; and to work together to foster the education of students (AACU, 2006).

In Nigeria Educational freedom and responsibilities enhance professionalism, specialization and abilities but requires proper allocation of responsibility.
This paper proposes the use of intelligent agent technology for academic responsibility assignment.
The proposed system should be able to perform the following task:

a. The lecturer agent will receive input from the Head Of Department (HOD) in term of 'departmental responsibility' and maximum work load. Responsibility is job description to perform and maximum work load is the maximum job description a lecturer is prepared to accept.
b. The lecturer agent will also be able to accept responsibilities on behalf of a lecturer, and give responsibilities and maximum work load.
c. The HOD agent will be able to update its catalogue with the departmental responsibilities available and the maximum workload a lecturer can assume.

d. The HOD agent will also be able to show the lecturer a list of available responsibility.

Formal Specification of the Intelligent Educational (e-educational) agent is the main focus of this paper which enhances responsibilities allocations within departments of a faculty within a University.

MATERIALS AND METHOD

Zed notations (Z) is the main ingredient used in this paper for the development of the educational agent that can allocate responsibilities on behalf of its HOD and the lecturer agent can accept these responsibilities based on specification, qualification and abilities.

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 prospective, and qualities of good specifications have been discussed (Sannella, 1988 and Spivey, 1992). Z-notation uses mathematical notation to describe in a precise way the properties a software system must possess, without unduly constraining the way in which these properties are achieved (Spivey 1998). Formal specification (Mathematical notation or Z) uses mathematical data types to model data in a system and achieve it underlining objectives. These data types are not oriented towards computer representation, but they obey a rich collection of mathematical laws which make it possible to reason effectively about the way a specified system will behave. We use the notation of *predicate logic* to describe abstractly the effect of each operation of our system, again in a way that enables us to reason about their behavior.

The other main ingredient in Z is a way of decomposing a specification into small pieces called *Schemas*. By splitting the specification into schemas, we can present it piece by piece. Each piece can be linked with a commentary which explains informally the significance of the formal mathematics. In Z, schemas are used to describe both static and dynamic aspects of a system (Spivey 1998).

The static aspects includes
a. the state it can occupy;
b. the invariant (quantity that is unchanged by a set of mathematical operation) relationship that are maintained as the system moves from states to state.

The dynamic aspect Includes:
a. the operation aspect that are possible;
b. the relationship between their input and output;
c. the changes of state that happen.

RESULTS

The following are some of the basic types in Z
{CHAR, STRING, CURRENCY, QUERY, OBJECT, COMPONENTS, BOOLEAN:: = TRUE/FALSE, DATA and OBJECT}

Every Lecturer is authenticated using his username/ID and password on the system

Lecturer

Lecturer_name/ID: seq CHAR
Lecturer_password: seq CHAR

Lecturer: $\mathbb{P}$ Lecturer
Access! : Boolean Lecturer

---

lecturer$\in$lecturer.access! = accepted) V (lecturer$\notin$lecturer.access $\neq$ accepted)

Figure 1: Lecturer Schema

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

AgentDirectory

Agent: ℙ AGENT; Responsibilities: ℙ RESPONSIBILITY
Directory: AGENT → RESPONSIBILITY

Agents = dom directory

Figure 2: AgentDirectory Schema

The directory lists all agents and responsibilities they provide and empower other agents looking for a precise responsibility to find the agent.

RegisterAgent

Δ AgentDirectory
agent? : AGENT
responsibility? : RESPONSIBILITY
report! : REPORT

(agent? ∉ agent ∧
AgentDirectory′ = AgentDirectory U {agent? → resposibility?}∧ report! = ok) ∨
(agent? ∉agent∧AgentDirectory′ = AgentDirectory∧ Report! = already_known)

Figure 3:RegisterAgent Schema

The directory braces the facility to register an agent given 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 responsibility they render.

FindResponsibility

Ξ AgentDirectory
Responsibility? : RESPONSIBILITY
Agentlist?ℙ AGENT

(agentlist! = {agent : agents / directory (agent) = responsibility? } ∨ (agent ∉ agents ∧ report! = not_known)

Figure 4:FindResponsibility Schema.

The Findresponsibility function obtains a responsibility type as an argument and returns all the agents who offer the service in a service list.

AlreadyExistingAgent

Ξ AgentDirectory
agent? : AGENT
result! : REPORT

agent? ∈ Agents
result! = already_known

Figure 5: AlreadyExistingAgent Schema

ResponsibilityNotAvailable

Ξ AgentDirectory
agents: ℙ AGENT
report! : REPORT
service? : RESPONSIBILITY

∀agent ∈agents.directory (agent) ≠ Responsibility
report! = not_known

Figure 6: ResponsibilityNotAvailable Schema

The schema in Figure 6 shows that error which occurs when an agent requests for an agent list for a particular responsibility, 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 responsibility in Figure 7.

Initialize_Directory

AgentDirectory

Responsibilitys: ℙ RESPOSIBILITY

agents ℙ AGENT

agent = ∅

responsibility = ∅

Figure 7: Initialize_Directory Schema

For online department purpose, each agent carries a department list of responsibility to be assigned and the maximum workload assigned.

DepartmentalResponsibilityList

responsibility: WORKLOAD

Maxresponsibility:ℙworkload

Departmentalresponsibilitylist: RESPONSIBILITY → workload

responsibility dom (DepartmentResponsibilitylist)

Figure 8: DepartmentalresponsibilityList Schema

The function *departmentalresponsibility* list in Figure 8, links each responsibility in the list to the maximum workload.

AddResponsibility

Δ DepartmentalResponsibilityList

workload? : WORKLOAD

Maxworkload? : WORKLOAD

Report! : REPORT

(resposibility∉responsibilities∧DepartmentalresponsibilityList′ = DeparmentalResponsibilityList U { responsibility → maxworkload}∨ Report! = known)

Figure 9: AddResponsibility Schema.

Responsibility can be added to the DepartmentalResponsibilityList with their corresponding maximum workload.

Removeresponsibility

Δ DepartmentalResponsibilityList

Responsibility? : RESPONSIBILITY

Report! : REPORT

(responsibility∉responsibilities∧DepartmentalResponsibilityList′ = DepartmentresponsibilitiesList / {responsibility → maxworkload} ∨

(responsibility? ∈ responsibilities∧ report! = known)

Figure 10: RemoveResponsibility Schema

Note that a DepartmentalResponsibilitieslist could be deleted that is, all the responsibilities 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 task on behalf of the agent sending the message. The action could be to check the catalogue for a particular responsibility and to return its workload 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 suggest a responsibility assignment pattern, if the responsibility is below the maximum workload limit otherwise 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, and the message type.

Message
From, To: AGENT
Date: DATE
Message: STRING
Msg? : MSG
Language: LANGUAGE
Msg ID: $\mathbb{Z}$
$3 \leq \#\text{msg ID} \leq \text{ID}$

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: $\mathbb{P}$ message
queue: $\mathbb{P}$ message
in = queue

Figure 12: Mailbox Schema

SendMessage
Δ Mailbox
out? = message
$in' = in \; ; \; out' = out \frown out?$

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
$In' = in \frown in? \; ; \; \#in = \#in + 1$

Figure 14: ReceiveMessage Schema

The HOD agent carries a catalogue of responsibilities for allocation and their workloads. The schema, AgentCatalogue (Figure 15), represents the HOD agent's catalogue.

AgentCatalogue
Responsibilities: $\mathbb{P}$ RESPONSIBILITIES
workloads: $\mathbb{P}$ WORKLOADS
Catalogue: RESPONSIBILITIES $\rightarrow$ WORKLOADS
Responsibilities = dom catalogue

Figure 15: AgentCatalogue Schema

Responsibilities are added to the catalogue by the HOD agent (Figure 16)and, are removed when they have been assigned (allocated) (Figure 17).

AddResponsibilitiesToCatalogue

Δ AgentCatalogue
Responsibilities? : RESPONSIBILITIES
Workload: WORKLOAD

(Responsibility$\notin$Responsibilities $\wedge$AgentCatalogue′ = AgentCatalogue$\wedge$ Report! $\neq$ already_known)

Figure 16: AddResponsibilitiesToCatalogue Schema

RemoveResponsibilitiesFromCatalogue

Δ AgentCatalogue
Responsibility? : RESPONSIBILITY

(Responsibility$\in$ responsibilities $\wedge$AgentCatalogue′ = AgentCatalogue$\wedge$ Report! = already_known)

Figure 17: RemoveResponsibilitiesFromCatalogue Schema

DISCUSSION

The case study of the intelligent educational agent that comprises Microsoft Windows NT operating system as platform, Structured Query Language server database as backend engine, Visual BASIC and Telescript commands infrastructure as frontend engine has been carried out. 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 educational agent system. Telescript Go and send command is used to transport the educational agent to the destination specified by the Ticket object. Telescript Self-command is promoting autonomy of the agent. The agent is able to create copies of itself. Telescript meet command is allowing the agent to interact with other agents at a *Meeting Place*. The agents only interact 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 lecturer agent and HOD agent. Now we can update the catalogue in the HOD agent by adding new available standard to it, that is, responsibilities and their workloads, which is displayed and then clicking “Add” to add a new ones. The HOD agents continuously wait for requests from lecturer agents. When asked to provide an offer for an item they check if the requested responsibility is in their catalogue and in this case reply with the workload, otherwise, they refuse. When they receive an assignment (allocation) order, they serve it and remove requested responsibility from their catalogue.

On the lecturer agent GUI we specify the list of responsibilities we want and how much time he is willing to spend for each of these responsibilities. As soon as an allocation is received, the lecturer agent accepts it based on the workload. If HOD agent provides more than enough allocation to a lecturer agent, then the lecturer agent accepts the best allocation fitting his experience. Having been assigned responsibility, the lecturer agent terminates, and a new search can be kicked off for another responsibility being added to the list.

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

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

a. The intelligent educational agent is able to assign responsibility on behalf of the HOD.
b. The intelligent agent system when implemented will save users (HOD and lecturer) time, as other jobs can be done by the user while the agent searches the web. Ordinarily, 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, worth mentioning, is the fact that some sites are designed to keep agents out and so, the lecturer agent is not assured of actually getting the "best" result on such site.

CONCLUSION

The impact of good intelligent agent systems in 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 agents can compare responsibilities and chose the best responsibility according to the lecturer's criteria. They detect changes in the lecturer's interests and alter to reflect those now facets. They can also monitor new responsible overtime and add it to the query if necessary. The query can be refined further. The agent activities takes place at the background while the lecturer does more important tasks like reading what has been retrieved, interpreting it, and acting upon result.

REFERENCE

AACU: Association of American Colleges and Universities (2006), "Academic Freedom and Educational Responsibility", retrieved online from www.aacu.org

Bjorn C. T. (1996), "Intelligent Agents and Conceptual Modeling", Trondheim University Press, Norway

John W. K. (2005), "What is a Software Agent?" retrieved online fromhttp:// agtivity.com/agdef.htm

Maes, P., (1994), "Agents that Reduce Work Load and Information Overload", Communications of the ACM, 37 (7): 31 – 40.

Sannella D., (1988), "A Survey of formal software development methods", appeared in Software Engineering: A European Perspective, A. McGettrick and R. Thayer (eds.), IEEE Computer Society Press, pp 281-297, 1993.

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, United Kingdom

Stuart R., and Norvig P. (1995), "Artificial Intelligence: A Modern Approach", Prentice Hall (UK) International.