A Collaborative Agent Framework

Mohd Hazli b. Mohamed Zabil
College of Information Technology,
Universiti Tenaga Nasional
Km 7, Jalan Kajang-Puchong,
43009 Kajang, Selangor
Tel: 603-89212020
hazli@uniten.edu.my

Mohd Sharifuddin Ahmad
Alan Cheah Kah Hoe
College of Information Technology,
Universiti Tenaga Nasional
Km 7, Jalan Kajang-Puchong,
43009 Kajang, Selangor
Tel: 603-89212020

Azhana Ahmad
Nazri Abdullah
College of Information Technology,
Universiti Tenaga Nasional
Km 7, Jalan Kajang-Puchong,
43009 Kajang, Selangor
Tel: 603-89212020

{sharif, alancheah}@uniten.edu.my {azhana, anazri}@uniten.edu.my

ABSTRACT
In a manual, non-mechanistic, face-to-face collaborative interactions, the tasks usually involve ad-hoc submissions (and consequential receptions) of documents. In a semi-mechanistic setting, some submission systems, e.g. an email system with (or without) attachments, animate the collaborative process. However, such system do not consider the role actors urgency and constraints in handling the tasks, e.g. the deadlines and types of documents for the tasks, which are not entrenched within the system’s parameters. Hence such system could not fully assist the role actors in performing the tasks effectively within the stipulated time, thus causing unnecessary delays in accomplishing the objectives.

We develop a slightly improvised collaborative framework, which entails a more systematic, time-based and event-based logic, involving the use of constraints via a central database. We introduce the concept of a role agent, which possesses a set of specific tasks to perform, the outcome of interactions between such agents generates a coherent set of coordinated actions. The side effect of these actions is the achievement of objectives as a consequence of the mutual fulfillment of individual roles. We attempt to animate such collaborative interactions in an examination preparation and moderation process of the College of Information Technology, UNITEN.

Categories and Subject Descriptors
1.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence – Intelligent Agents, Multiagent Systems.

General Terms
Algorithms, Design, Theory.

Keywords
Software Agent, Multiagent System, Collaborative Interaction.

1. INTRODUCTION
The research in software agents have progressed over more than a decade emerging from research in distributed artificial intelligence (DAI) and distributed computing. The prime motivation for developing software agents stems from the demand for programs that can interoperate by exchanging information and services with other program, thereby solving problems that cannot be solved in isolation [2].

The word agent can be defined as one that is authorized to act for another. Agents possess the characteristics of delegacy, competency, and amenability [7]. In the context of computer science, there are many definitions given for software agent. Among others, an agent refers to a component of software and/or hardware which is capable of acting exactly in order to accomplish tasks on behalf of its user [12]. Wooldridge and Jennings [14] define the term agent as hardware- or software-based computer with the agent attributes. These attributes differentiate a software agent with other agent-like software. The attributes are autonomy, social ability, reactivity and pro-activeness. On top of that, a stronger notion of agency should carry extra attributes, which are more human-like such as mental states (belief, desire and intention), emotions, mobility, veracity, benevolence and rational.

A multi-agent system is a system which needs more than one agent to collaborate and complete tasks needed by the system. Although most multi-agent systems are complex and complicated, not all complex systems need to be multi-agent systems. A multi-agent system is needed when there are different users with different roles and goals. All of them need to collaborate between each other in order to achieve the system’s goal.

Collaboration is a joint-work by two or more parties, to achieve a goal. In this project, there are multiple agents with different roles, each has its own goal. The collaborative agent, is not just an interaction between agents in the system. It is some kind of joint-working activity by two or more agents in achieving a goal. Grosz [10] lists four main characteristics for a collaborative system. First, it is not a kind of master-servant relationship. Second, most collaborative situations involve agents who have different beliefs and capabilities. Third, multi-agent planning entails collaboration in both planning and acting. And lastly, collaborative plans are not simply the sum of individual plans.

2. PROBLEMS IN EXISTING PROCESS
In the current process of examination paper preparation and moderation in the College of IT, there are problems arising from possibility such as deadlines being exceeded by moderators and lecturers. Such occurrence results in process inefficiencies as consequences of loss of time and manpower. When the deadlines have passed, the moderators have limited time to evaluate and assess the papers, which thus compromises the quality of examination papers. Such loss of time is cascaded down to other subsequent tasks that need to be completed by other lecturers.
Another problem with the current process is the lack of urgencies in preparing and moderating the examination papers. Due to a wide range of tasks a lecturer has to perform, and the ample time given to them to prepare and moderate examination papers, there is the tendency to put off such task to the last minute. Without an efficient monitoring system, by which moderators and lecturers are alerted to complete and submit their papers, delays could be entrenched in the current manual system.

We attempt to resolve these issues by developing an improvised collaborative framework, which entails a more systematic, time-based and event-based logic, involving the use of constraints via a central database. We introduce the concept of a role agent, which possesses a set of specific tasks to perform, the outcome of interactions between such agents generates a coherent set of coordinated actions. The side effect of these actions is the achievement of objectives as a consequence of the mutual fulfillment of individual roles.

3. DEVELOPMENT APPROACH
We propose to develop such framework based on the methodology discussed in [2].

3.1 Initiate a Case Study
An appropriate launching point for synthesizing a collaborative framework is to initiate a test case involving the relevant roles of a lecturer in the College of IT.

The objective is to determine the lecturer’s tasks in preparing the final examination papers and moderating papers from other lecturers. We are interested, in particular, in the interaction attributes between lecturers.

3.2 Synthesize a Collaborative Framework
We use the results from the study in 3.1 above to develop a logical model relating to the lecturer’s role. We conceptualize the model by iteratively considering the relationships the role has with the constraints, information artifacts and the domain ontology. The initial outcome of this process is a script, representing an agent, which executes the role on behalf of the lecturer.

In this project, we model and design the agents based on [1] and [2], by formulating an agent’s action script.

3.3 Formulate Constraints
We use SICStus Prolog [13] as an engine to constrain task executions based on two parameters, temporal and artefactual. The general premise of our constraint formulation is that constraints are satisfied when tasks are performed successfully within the stipulated time and information artifacts are of the right attributes. If all constraints have been successfully satisfied, we say that the tasks have been successfully performed.

4. MODELLING A ROLE AGENT
We describe here the initial theory of modeling the role agent, which is based on [1] and [2].

4.1 Roles and Tasks
We argue that a role consists of one or more tasks and that each task is planned by the role actor as a set of actions known as a script. Each script is assumed to be associated with some sort of constraints. The actions in the script could operate upon resources, the names of which are assigned by the role actor. The names of these resources together with the constraints constitute the ontology of the role actor. Further elaboration of an ontology can be found in [11].

4.2 Action Primitives
In attempting to develop the action script for a role actor, we propose three action primitives – input, output and new. To implement the script, we concoct the concept of an agent, which represents the role actor. The agent has a workspace to which resources will be saved and in which new resources will be created. The resources will then be stored at a location, possibly a database, Db. For each action, we propose a duration of the action represented by a start time, ST and an end time, ET. The syntaxes (represented by Prolog clauses) and semantics for such action primitives are as follows [1][2]:

- input(Db, Loc(R), L, ST, ET).

  The input action primitive copies a resource R, in location Loc, of a database Db, to location L, in a local workspace. The action starts at time ST and ends at time ET.

- new(R, Rs, ST, ET).

  The new primitive uses the input resource R in the local workspace to create a new resource Rs at time ST and ends at time ET.

- output(Db, L(Rs), Loc, ST, ET).

  In contrast to the input primitive, the output action copies Rs from location L in the local workspace to a location Loc in Db at time ST and ends at time ET.

Prior to the execution of the action script, the script contains variables, which, unlike the normal Prolog variables, are ground atoms. These variables must be defined in the agent’s ontology and will be bound to a value as the scripts are gradually implemented.

4.3 Control Primitives
In a real situation, a role actor attempts one or more alternatives based on a certain condition. A more flexible and robust script construct is required to provide the role actor with such alternatives. We define control action primitives to cater for conditional and non-conditional situations [2].
To express an alternative course of action, an algorithm similar to the if-then-else construct provides a familiar operation of choosing one out of two alternatives. We express such operation in a choose primitive, which selects one out of two alternative courses of actions. We define the primitive as follows:

- \text{choose}\left(\text{Condition\_Call, Block\_Id1, Block\_Id2, ST, ET}\right).

where \text{Condition\_Call} is a user-defined call to a condition to be tested, \text{Block\_Id1} and \text{Block\_Id2} are block identifiers, which point to target blocks, and \text{ST} and \text{ET} are the start and end times respectively.

When the condition is tested, the result of the test determines either \text{Block\_Id1} or \text{Block\_Id2} is executed. The script interpreter can be written such that if the test result is true, \text{Block\_Id1} is executed, otherwise it executes \text{Block\_Id2}. The block execution always starts with the first action in the script. The condition usually relates to the bound (or unbound) values of the script variables and/or ground atoms. When either execution of \text{Block\_Id1} or \text{Block\_Id2} has been completed, control returns to the next action in the calling block.

In some cases, execution may end without returning to the calling block. To express a situation in which the control proceeds to a branch if the test condition is true but remains on the current block otherwise, the action is written as:

- \text{choose}\left(\text{Condition\_Call, Block\_Id1, null, ST, ET}\right).

The \text{choose} action renders a script as a tree of actions branching to another action at the \text{choose} nodes. For this reason, an identification scheme is needed to match the address of the target block with that returned by the test result.

To give the script structure even more expressiveness by executing a block of script from within another block, we propose a \text{goto} primitive:

- \text{goto}(\text{Block, ST, ET}).

where \text{Block} represents a destination block having several actions, and \text{ST} and \text{ET} are the start and end times respectively.

5. THE APPLICATION DOMAIN
Our application domain is the process of preparing and moderating examination papers at the College of IT. The process would make a good case to test our framework because it entails tasks that are collaborative in nature where exchanges of documents are performed in a pre-determined schedule. The tasks are distributed within a community of lecturers with different roles and where the drudgery of keeping tabs of the tasks could be delegated to software agents. Figure 1 shows the process flowchart [6].

The process begins when the Examination Committee (EC) Secretary announces the tasks and deadlines for preparing and moderating examination papers. A sample of the tasks and their corresponding deadlines are shown in Table I.

---

**Table I  Tasks and Deadlines for Examination Paper Preparation and Moderation**

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Week No. (Deadlines)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set A to be submitted to the respective moderators</td>
<td>10 (12 Feb)</td>
</tr>
<tr>
<td>1st moderation</td>
<td>10 – 11 (12 – 25 Feb)</td>
</tr>
<tr>
<td>2nd moderation</td>
<td>12 – 13 (26 Feb – 11 Mar)</td>
</tr>
<tr>
<td>Set B to be submitted to EC</td>
<td>14 (12 Mar)</td>
</tr>
</tbody>
</table>

All lecturers will then start to prepare their examination papers. Upon completing the preparation, they will submit their papers to the respective moderators for moderation. If the moderator detect any discrepancies, the moderator will return the papers to the lecturer for corrections. Two cycles of moderation are conducted to ensure that all papers are fully moderated, checked and re-checked to ensure their high quality.

Upon completion of the moderation process, the moderators will return the moderated papers to the lecturers, who will forward the papers to the EC. The EC will then perform another cycle of checking. If there are corrections to be made, the papers will be returned to the lecturers for further corrections. Otherwise the EC Secretary prepares the papers for printing and storage.
From Figure 1, we identify three roles for which agents could be represented: Lecturer, Moderator, and EC Secretary. For each role, we determine the inputs that are required to produce the output. For example, for a lecturer to prepare an examination paper, he/she would need the following inputs:

- presentation materials of the subject taught
- course syllabus
- last semester’s examination paper
- second last semester’s examination paper

To produce the output, the lecturer needs to create two new documents, which are:

- examination paper, and
- solutions with the marking scheme.

Having analyzed the process, we develop a logical model of the lecturer’s process by applying the action primitives. Applying the input primitives for the input documents, we write:

\[
\text{new(new([lsub(f1), ldoc(f2), ldoc1(f3), ldoc2(f4), lex1(exqs)], lex1(exqs), s5, e5)), s6, e6).}
\]

We make the following assumptions about the input parameters:

(i) \(db244\) is a database, which stores all the documents accessible to the lecturer.

(ii) \(ldb11, ldb12, ldb13, ldb14\) represent the locations of the presentation materials for the subject \((f1)\), the course syllabus \((f2)\), the last semester’s examination paper \((f3)\), and the second last semester’s examination paper \((f4)\) respectively, in the database \(db244\).

(iii) \(lsub, ldoc1, ldoc2\) and \(ldoc3\) are the documents corresponding location in the lecturer’s local workspace.

After having saved the input documents, the lecturer then starts to create the examination paper. We represent such action by applying the new primitive as follows:

\[
\text{new(new([lsub(f1), ldoc1(f2), ldoc2(f3), ldoc3(f4)], lex1(exqs), s5, e5)), s6, e6).}
\]

By referring to all the input documents, the lecturer creates a new examination paper, \(exqs\), in a new location, \(lex1\), in his/her local workspace. Similarly, he/she creates another new document, which is the solutions with the marking scheme, \(exms\), in another new location, \(lex2\). We represent such action as:

\[
\text{new(new([lsub(f1), ldoc1(f2), ldoc2(f3), ldoc3(f4)], lex1(exqs), s5, e5)), s6, e6).}
\]

Upon completing the creation of the new documents, the lecturer then submits the documents to the moderator. In a similar fashion, we represent such action by applying the output primitive as follows:

\[
\text{output(output(db245, lex1(exqs), ldb21, s7, e7)), output(db245, lex2(exms), ldb22, s8, e8).}
\]

We assume the following:

(i) \(db245\) is a database, which stores all examination papers accessible to the moderators.

(ii) \(ldb21\) and \(ldb22\) are the documents corresponding location in the database \(db245\).

Subsequently, we compile the lecturer’s actions in preparing the examination paper to produce the actions representation as a lecturer agent’s script:

**Lecturer agent:**

\[
\text{input(db244, ldb11(f1), lsub, s1, e1)},
\]

\[
\text{input(db244, ldb12(f2), ldoc1, s2, e2)},
\]

\[
\text{input(db244, ldb13(f3), ldoc2, s3, e3)},
\]

\[
\text{input(db244, ldb14(f4), ldoc3, s4, e4)}.
\]

\[
\text{new(new([lsub(f1), ldoc1(f2), ldoc2(f3), ldoc3(f4)], lex1(exqs), s5, e5)), s6, e6).}
\]

\[
\text{output(output(db245, lex1(exqs), lmod11, s7, e7), output(db245, lex2(exms), lmod12, s8, e8))}.
\]

Similarly, by repeating the analysis for the other role actors, the Moderator and the EC Secretary, we produce their corresponding agent scripts:

**Moderator agent:**

\[
\text{input(db245, lmod11(exqs), lmdex1, s9, e9)},
\]

\[
\text{input(db245, lmod12(exms), lmdex2, s9a, e9a)},
\]

\[
\text{input(db244, ldb13(f3), lmdoc3, s10, e10)},
\]

\[
\text{input(db244, ldb14(f4), lmdoc4, s11, e11)}.
\]

\[
\text{input(db243, ldb21(mdfrm), lmdfrm, s12, e12)}.
\]

\[
\text{new(new([lmdex1(exqs)], lmdex3(cexqs), s13, e13)), new([lmdex1(exqs)], lmdex3(cexqs), s14, e14)}.
\]

\[
\text{output(output(db246, ldb23(cexms), lmdex3, s15, e15)), output(db246, ldb24(cexms), lmdex4, s16, e16)}.
\]

**EC Secretary agent:**

\[
\text{input(db246, lmdex1(cexqs), lsec11, s17, e17)},
\]

\[
\text{input(db246, lmdex2(cexms), lsec12, s18, e18)}.
\]

Our analysis of the scripts shows that the outcome of a collaborative interaction between these agents, assuming all tasks and deliverables are completed within the stipulated times, is a set of documents representing the examination question paper and the solutions with the marking scheme, ready for printing.

### 6. FURTHER WORK

#### 6.1 Incorporating Control Primitives

The above scripts are skeletal representations of agents collaborating in an ideal and rigid situations. To implement a robust and flexible actions based on the conditions of the domain, we will incorporate the control primitives.

For example, if a moderator rejects an examination paper, the lecturer agent should be able to discriminate a new and a rejected examination paper so that, based on this condition, a proper sequence of action is taken to remedy the situation. For this purpose, a conditional script construct utilizing the choose or goto control primitive would be required.
6.2 Formulating Constraints

A constraint is a logical relation among several variables, each taking a value in a given domain. The constraint restricts the possible values that variables can take, thus imparting partial information concerning the variables of interest [2].

Researchers have developed logical models to resolve problems related to events involving time and interval. Specific approaches have been developed to handle quantitative constraints as well as qualitative ones. Quantitative constraints handle problems that provide definitive solutions in time point and interval events whereas qualitative constraints are concerned with the relative time of events.

In this research we are concerned with the qualitative constraints, whereby based on the prevailing conditions of the variables in the domain, specific dates must be determined to advise the role actors to respond to critical actions before deadlines are exceeded. Delays in performing the actions will progressively reduce the time to respond until a total failure in performance is inevitable.

We will use the SICStus Prolog’s built-in constraint engine to formulate a set of constraints, primarily the deadlines, imposed on the actions of the role actors.

6.3 The Java Agent DEvelopment Framework

The agent-based collaborative framework developed in [2] was based on a logic programming paradigm. We are exploring the possibility of running the role agents in other agent-based platforms to avoid the drudgery of writing dedicated codes for agent interactions and administration. While there are a handful of agent-based frameworks available in the public domain, we are looking at those that are FIPA-compliant [9]. One such framework is the Java Agent Development Framework or JADE [5].

JADE is a software framework fully implemented in Java. It simplifies the implementation of multi-agent systems through a middleware that claims to comply with the FIPA specifications and through a set of tools that supports the debugging and deployment phase. The agent platform can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI.

JADE facilitates the development of multi-agent systems by offering:

- A runtime environment where JADE agents can “live” and that must be active on a given host before one or more agents can be executed on that host.
- A library of classes that programmers have to/can use (directly or by specializing them) to develop their agents.
- A suite of graphical tools that allows administrating and monitoring the activity of running agents.

Each running instance of the JADE runtime environment is called a Container as it can contain several agents. The set of active containers is called a Platform. A single special Main Container must always be active in a platform and all other containers register with it as soon as they start. It follows that the first container to start in a platform must be a main container while all other containers must be “normal” (i.e. non-main) containers and must “be told” where to find (host and port) their main container (i.e. the main container to register with).

The communication architecture offers flexible and efficient messaging. The full FIPA communication model has been implemented and its components have been clearly distinguished and fully integrated: interaction protocols, envelope, ACL, content languages, encoding schemes, ontologies and, transport protocols.

Figure 2 illustrates the above concepts through a sample scenario showing two JADE platforms composed of 3 and 1 container respectively.

![Figure 2 JADE’s Containers and Platforms](image)

From our analysis of the JADE framework, we deduce that the framework would offer a comprehensive suite of tools to implement the agents as well as to incorporate future communication capabilities for collaboration between agents.

7. CONCLUSION

In this paper we discussed the preliminary results of our research in incorporating an agent-based framework in a collaborative process. To test our case, we chose the collaborative process of examination paper preparation and moderation in the College of IT. Such process has the attributes and characteristics that are feasible for an agent-based framework.

We developed the action primitives for agents to react and coordinate in the preparation and moderation process. Although the agents do not perform the main tasks, they provide the mechanism to collaborate with other agents to ensure that a coordinated actions is generated to achieve the desired objective. We have shown that the agent scripts that represent the role actors’ actions produced the desired outcome. We will develop the scripts further to incorporate controlled actions based on prevailing domain conditions.

8. REFERENCES


