Mobile Agent based realization of a distance evaluation system

Vikram Jamwal and Sridhar Iyer

KR School of Information Technology, IIT Bombay
India, 400 076
{vikram, sri}@it.iitb.ac.in

Abstract

The growth of the Internet has led to new avenues for distance education. A crucial factor for the success of distance education is effective mechanisms for distance evaluation. Existing computer based evaluation mechanisms, such as Web Based Testing, rely principally on the client-server model. Such mechanisms usually do not scale well and also do not fully support features like: evaluation of subjective questions, delivery of dynamic content, and off-line examinations. These features are extremely desirable for distance evaluation and there is a need for alternate ways of designing such applications.

We propose the use of Mobile Agents for effective structuring of distance evaluation. Mobile Agents are autonomous and dynamic entities that can migrate between various nodes in the network. They offer many advantages over traditional design methodologies like: reduction in network load, overcoming network latency and disconnected operations.

We have designed and implemented MADE, a Mobile Agents based system for Distance Evaluation of students who may be spread over large areas. MADE aims to map closely to real world examination scenarios and addresses the full gamut of the examination process, viz., paper setting, distribution and testing, evaluation and result-compilation. In this paper we present our experiences in the development of MADE and show how Mobile Agents can be leveraged to effectively structure such large-scale applications.

1. Introduction

The widespread penetration of the Internet has made it possible to impart education on a larger scale. This has resulted in new models for knowledge dissemination by universities and other organizations. Distance evaluation (DE) of students constitutes a crucial factor for the success of distance education initiatives.

1.1. DE Application scenario

We consider an examination scenario where a large number of students are e-evaluated concurrently. A typical large-scale examination process involves the following stages: (i) preparation of question papers by gathering inputs from various paper-setters who may work at their respective remote locations, (ii) dispatch of question papers to the examination centers and distribution to the enrolled students, (iii) collection of answer papers and their dispatch to the evaluation center, (iv) evaluation of answer papers by the designated evaluators, and (v) compilation and publication of the results.

1.2. Traditional techniques for DE

Most of the present day Internet based evaluations, e.g. [1], are web centric and employ the client-server paradigm. They extend the principles of Computer Based Testing (CBT) to evaluation across wide area networks. CBT has been deployed for examinations like Graduate Record Examinations (www.gre.org) and allows for asynchronous and round-the-year examinations [2]. However, it depends upon a local database of the questions (typically on a local area network) and does not scale well for remote testing.

In Web Based Testing (WBT) [3], on the client side the students download a questionnaire as a web page and submit the answers back to the server. The server evaluates the answers and returns the results to the client. Java Applets and scripting languages like Java Script etc. are the frequently used techniques to enable front-end client processing. Common Gateway Interface (CGI) scripts or Java Servlets are the most often used techniques for server side processing.

1.3. Desirable extensions

The above techniques use the client-server paradigm and as such are susceptible to problems due to varying network characteristics. They present scalability problems when a large number of users access the server simultaneously.

In addition there is a need to provide the following features:

- *Comprehensive solution*: Paper setting, distribution, collection, result compilation and publishing are important parts of the DE application and should be well integrated with each other and the rest of evaluation system.

- *Support for subjective questions*: Answers that involve written text or graphical schematics would normally require manual evaluation by one or more evaluators. The system should support a workflow of answer papers among these evaluators.
- *Delivery of dynamic content*: Questions may need to be presented to the students using dynamic content in the form of audio, video, multimedia etc. Sometimes it might also be necessary to send a tool (e.g. a compiler for client-side code compilation and testing) to the students.
- *Offline examinations/ operations*: Unreliable links, security and other reasons might require that students, paper-setters, and evaluators work offline for certain durations.
- *Support for push*: There are cases where pushing information to the users is a better alternative than the users pulling the information from the servers. E.g., such a need may arise when some run-time notices are to be communicated to the students.

Since the existing WBT mechanisms primarily use the client-server and pull model of distributing information, we feel that it would be cumbersome to extend them to provide the above features. Hence, there is a need for alternate mechanisms.

1.4. Our approach: using Mobile Agents

Over the past few years, the Mobile Agent paradigm has emerged as a new mechanism for structuring distributed applications. It promises to alleviate many of the shortcoming of the client-server approach [4,5,6]. Mobile Agent (MA) is an autonomous piece of software that can migrate between the various nodes of the network and can perform computations on behalf of the user [7]. Some of the benefits provided by MAs include reduction in network load, overcoming network latency and disconnected operations [8].

For our application, Mobile Agents seemed particularly useful because they map directly to real life situations, are dynamic autonomous entities, and can work in both push and pull modes. We have designed and implemented MADE, a Mobile Agent based system for distance evaluation. We used the Voyager framework [9] to implement our system.

In this paper, we present our experiences with the design and implementation of MADE. From our experience, we have identified some key characteristics/requirements of an application that determine its suitability for MA based design.

The organization of this paper is as follows. Section 2 provides the architecture of MADE. Sections 3 and 4 present the detailed design and implementation aspects of MADE respectively. Section 5 and 6 present advantages gained by using MAs for distance evaluation and gains for large-scale application-structuring, respectively. Section 7 concludes with a discussion on challenges that need to be tackled.

2. MADE architecture

MADE is a Mobile Agent based system for distance evaluation of students. It was designed with a view to map closely to the real world scenario. Other goals include automation and integration of the entire examination process, minimization of infrastructure requirements at different nodes, and ease of deployment and maintenance. In MADE we divide the examination process into three stages: (i) paper-setting, (ii) distribution and testing and (iii) evaluation and result compilation.

2.1. Paper setting

As shown in figure 1, the examination setting process takes place in a collaborative manner among the paper-setters who are at different remote locations. Install Agents are used to install the paper-setting application on each setter's machine (Step 1). Each setter prepares a partial question paper (Step 2). Fetch Agents are subsequently dispatched to collect these question papers (Step 3). The Paper Assembler node creates one/more comprehensive question paper from the partial papers (Step 4). One of this question papers is sent to the examination centers at the appropriate time (Step 5).

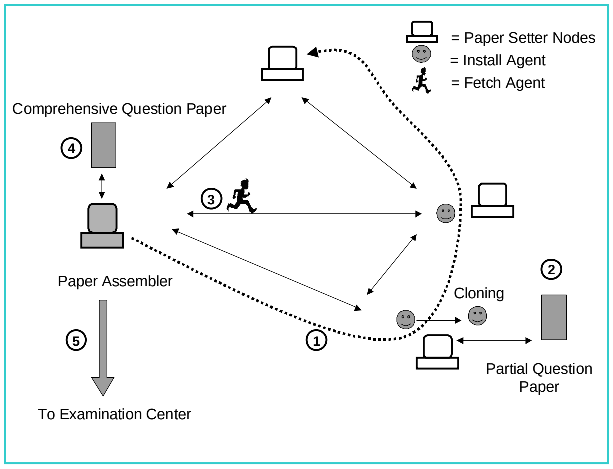


Figure 1. Paper setting scenario

2.2. Distribution and testing

As shown in figure 2, this stage involves sending the question paper to different centers, distribution to students and the collection of answer papers. The question paper is dispatched to the different examination centers with the help of Courier Agents (Step 1a, 1b). The Distribution Server at each center has a list of candidates enrolled for that center. It creates Question Agents (one per student) and dispatches them to each student node in the center (Step 2, 3). After the designated examination duration or when the student finishes, each Question Agent returns to the Distribution Server with the student's answers (Step 4). The Distribution Server now creates an Answer Agent for each answer-paper (Step 5), and sends it to the Evaluation Server.

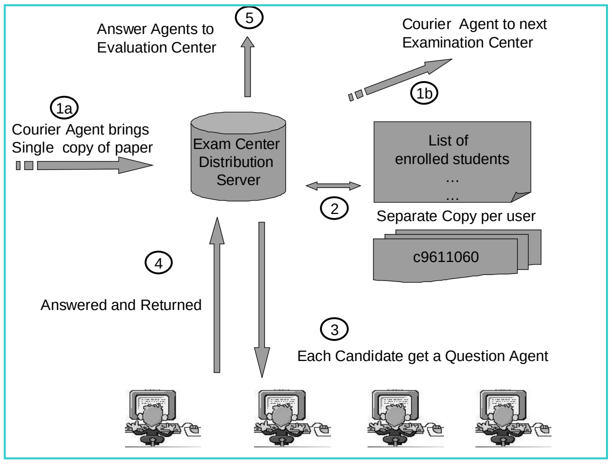

Answer Agents to Evaluation Center
5
Courier Agent to next Examination Center
1b
1a
Courier Agent brings Single copy of paper
Exam Center Distribution Server
List of enrolled students
…
…
2
Separate Copy per user
c9611060
4
Answered and Returned
3
Each Candidate get a Question Agent

Figure 2. Distribution and testing scenario

2.3. Evaluation and result compilation

As shown in figure 3, this stage involves evaluation of the answer papers, compilation of the results and their publication. When an Answer Agent reaches the Evaluation Server, it is supplied with an itinerary of evaluators (Step 1). The Answer Agent visits various evaluators, until all the answers are evaluated (Step 2). Finally the Answer Agent moves to the Publishing Server where it supplies its results (Step 3). The comprehensive results are then compiled and published (Step 4).

In the next sections we provide the detailed design and implementation aspects of MADE.

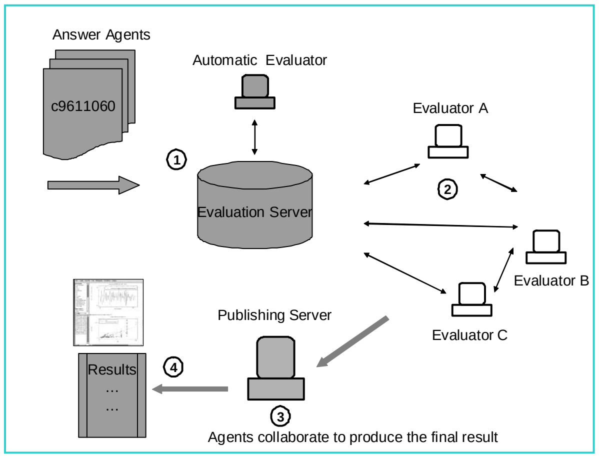


Figure 3. Evaluation and result compilation scenario

3. MADE: Detailed design

In this section, we present the design details in MADE in terms of Mobile Agents used, the main application components and their interactions.

3.1. Paper setting stage

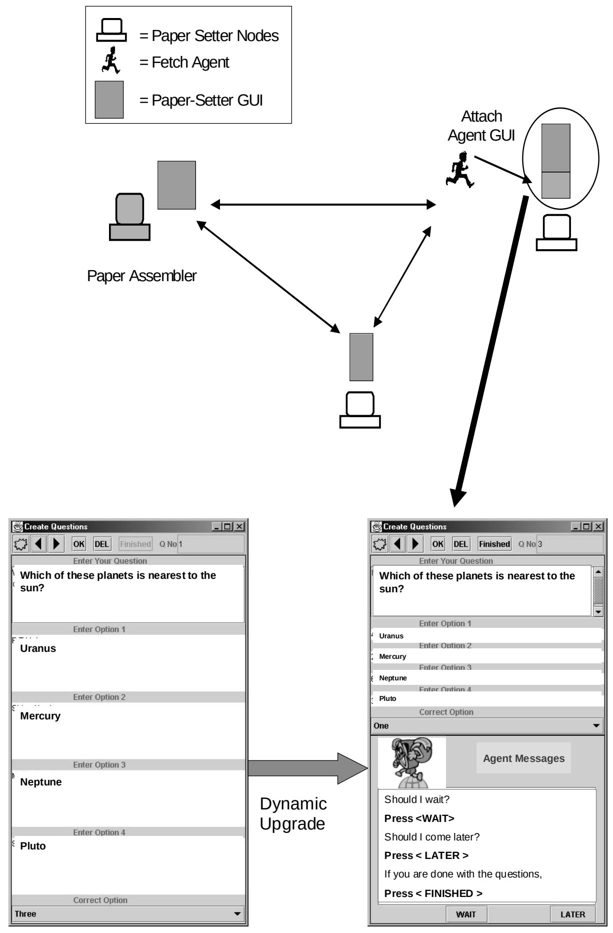
Paper setting involves: (i) the installation of application on the paper setter nodes, and (ii) subsequent fetching of the partial question papers from the paper-setters.

3.1.1. Application installation on paper-setter nodes. The paper-setting process is coordinated by a node termed Paper Assembler. This node is given a list of paper-setters for each subject.

For installing the paper-setting application at various nodes, the Launcher object at the Paper Assembler node instantiates and launches an Install Agent. This agent is supplied with an itinerary, which consists of the list of paper-setters that have to be visited. The Install Agent moves to the first paper-setter's machine and installs the Paper Setter Application. The Paper Setter Application registers itself with the Naming Service. Any object that wants to communicate with this application in future will use this reference. If more installations are to be done, the Install Agent clones itself and moves to the next paper-setter. In this way the application is installed on all the paper-setter machines.

3.1.2. Fetching of partial question paper. The paper-setters might prepare question paper over several days. When it is time to collect the question papers, the Launcher object instantiates a Fetch Agent. This agent can either be supplied with the full itinerary at the source or it can build its itinerary dynamically (the information of next paper-setter is available with the Naming Service at each paper-setter).

Upon arriving at a paper setter node, the Fetch Agent queries the Naming Service for a reference to the Paper Setter Application at that node. The Fetch Agent creates a Graphical User Interface (GUI) object and attaches it to the paper-setter's GUI, dynamically at runtime, as shown in figure 4. This interface prompts the paper-setter to submit her questions. Depending upon the response of paper-setter, the Fetch Agent may go into either of these states: *wait* (wait for the submission), or *deferred* (move to the new location and come back later). The third option is to *force-fetch* i.e. after the expiry of designated duration, the Fetch Agent can forcibly take away the partial question paper from the paper-setter.



Which of these planets is nearest to the sun?

Uranus

Mercury

Neptune

Pluto

Three

One

Should I wait?
Press <WAIT>
Should I come later?
Press < LATER >
If you are done with the questions,
Press < FINISHED >

Figure 4. Dynamic upgrade of paper-setter's application by the Fetch Agent

Once the Fetch Agent gets a partial question paper, it moves to the Paper Assembler to submit it. After this the Fetch Agent's itinerary consists of the remaining paper-setters. It keeps polling these paper-setters until all of them have submitted their questions. A summary of the components used in paper-setting stage is given in table 1 and the interactions between these components (using UML notation [10]) are shown in figure 5.

The Paper Assembler assembles the final question paper after it has received inputs from the various paper-setters. In the next stage, the question paper needs to be distributed to various examination centers.

Table 1. Main components used in paper setting

	Component	**Functionality**
1	Paper Assembler	Coordinates various paper-setters; assembles the final question paper after receiving inputs from the paper-setters
2	Launcher	Creates and launches the Install Agent and Fetch Agent; creates the Paper Assembler application
3	Install Agent	Installs paper setting application and interface on remote paper-setter nodes
4	Fetch Agent	Collects partial question papers from the paper-setter nodes
5	Paper Setter Application	Application used by the remote setter to set the question paper.

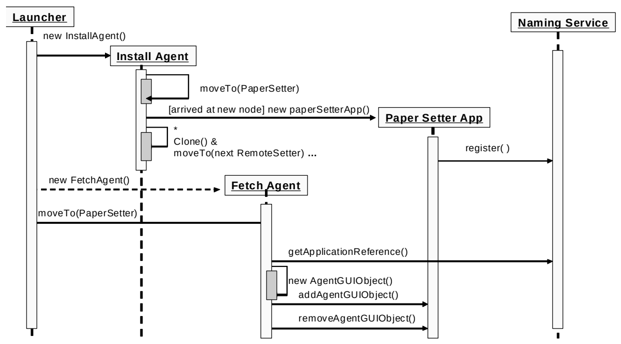


Figure 5. Component interactions in paper setting stage

3.2. Distribution and testing stage

This stage involves: (i) distribution of question papers to the examination centers, (ii) creation of Question Agents for testing the students and (iii) creation of Answer Agents from the students' responses.

3.2.1. Distribution of question papers. The Paper Assembler node creates and launches a Courier Agent after supplying it with the question-paper object and the itinerary of various examination centers. This agent carries only a single copy of a particular question paper. After supplying a copy of the question paper to the Distribution Server at an examination center, the Courier Agent moves on to the next location. Upon completion of the itinerary, it returns to the Paper Assembler and terminates. If the number of examination centers is large, more than one CA may be launched in parallel.

3.2.2. Creation of question agents and testing. The Distribution Server has a list of the students enrolled for that center. It maps each machine in the center to a student and instantiates one Question Agent per student. Once the Question Agent reaches the student's machine, the student can work (or can be made to work) offline for the duration of test. The Question Agent presents the questions to the students and records his answers. After the designated examination duration or when the student finishes, the Question Agent returns to the Distribution Server with the answers.

3.2.3. Creation of Answer Agents. The Distribution Server extracts answers from the Question Agent and creates an Answer Agent. The Answer Agent is later sent to the Evaluation Server.

Note that while the Question Agent itself could be sent to the Evaluation Server, we use a separate Answer Agent to ensure security and anonymity. For example, student machines may not be trusted hosts and the use of Question Agent hides information about the evaluation process from the student. Similarly, the Answer Agent hides student details from the evaluators.

A summary of the components used in the distribution and testing stage is given in table 2 and the interactions between these components are shown in figure 6.

Table 2. Main components used in distribution and testing

	Components	**Functionality**
1	*Courier Agent*	Delivers the question paper to the Distribution Servers of all examination centers.
2	*Distribution Server*	Creates Question Agents and sends them to the students in the center; creates Answer Agents and sends them to the Evaluation Server.
3	*Question Agent*	Presents the questions to a student and carries the answers back to the Distribution Server
4	*Answer Agent*	carries the answer paper of a student to the Evaluation Server

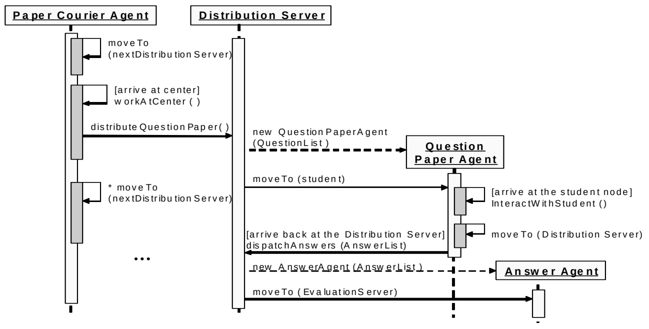

Paper Courier Agent
Distribution Server
moveTo (nextDistributionServer)
[arrive at center] workAtCenter ( )
distributeQuestionPaper( )
new QuestionPaperAgent (QuestionList)
Question Paper Agent
moveTo (student)
[arrive at the student node] InteractWithStudent ()
* moveTo (nextDistributionServer)
moveTo (Distribution Server)
[arrive back at the Distribution Server] dispatchAnswers (AnswerList)
...
new AnswerAgent (AnswerList)
Answer Agent
moveTo (EvaluationServer)

Figure 6. Component interactions in distribution and testing stage

3.3. Evaluation and result compilation stage

This stage involves: (i) evaluation of answer papers and (ii) compilation and publication of results.

3.3.1. Evaluation of Answer Papers. The Evaluation Server examines the *type* of answers that an Answer Agent is carrying to find our whether they can be machine evaluated or need manual evaluation.

Automatic Evaluations: Some questions, such as multiple-choice, may be automatically evaluated. The Answers Agent obtains a reference to an Automatic Evaluator from the Evaluation Server. The Answer Agent then moves to the Automatic Evaluator and requests for an evaluation. The Automatic Evaluator evaluates these answers and returns the scores to the Answer Agent.

Manual Evaluations: The Evaluation Server has a set of Evaluators for each question paper. It prepares an itinerary for each Answer Agent. The Answer Agent then moves to each Evaluator to get its answers evaluated. Once an Answer Agent reaches an Evaluator, it exports a GUI to the evaluator and prompts her to evaluate its answers. After getting itself evaluated or after a designated time, it moves on to the next Evaluator.

Initially the Answer Agent visits each Evaluator in the order specified in its itinerary. After the first round of evaluations, it visits them dynamically depending upon the questions that still need to be evaluated.

3.3.2. Publication of Results. When all the answers have been evaluated, the Answer Agent obtains a reference to the Publish Server from the Evaluation Server. It then moves to the Publish Server and supplies its scores. After all the Answer Agents have supplied their scores, the Publish Server compiles and publishes the final results.

A summary of the components used in the evaluation and result compilation stage is given in table 3 and the interactions between the components are shown in figure 7.

Table 3. Main components used in evaluation and result compilation

	Components	Functionality
1	*Answer Agent*	Carries student's answer paper; visits the Evaluators to get itself evaluated and supplies the final results to Publish Server
2	*Evaluation Server*	Coordinates the evaluation process. All the evaluators register themselves with this server. In addition it has reference to the Automatic Evaluator and the Publish Server
3	*Evaluator*	Evaluates answers; may be automatic / manual
4	*Publish Server*	Accepts individual results of each Answer Agent and then compiles and publishes the final results.

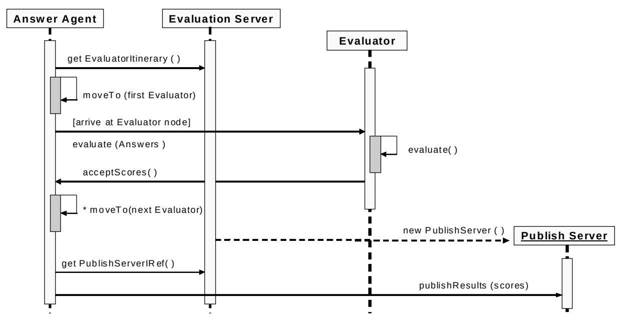


Figure 7. Component interactions in evaluation and result compilation stage

In the next section, we present some of the implementation aspects of MADE.

4. MADE: Implementation aspects

MADE has been implemented using Java as the development language and Voyager ORB [9] as the MA Framework. We chose Voyager mainly because it is a generalized platform for distributed object computing and allows easy integration of MAs with the rest of the application components.

The prototyping was carried out on Pentium III, 450 MHz workstations (running either on Windows2000 or Linux operating systems) on a campus network. Voyager ORB was installed on all of these machines. We simulated the set up for the examination process by using different ports on each machine for the different services.

A summary of the MAs used in MADE is given in table 4.

Table 4. Mobile Agents used in MADE

MA Type	No of Hops	Life-Duration.	No. of Instances	Itinerary
Install Agent	1 to no. of paper-setters	few months (paper setting duration)	1 to no. of setters (10s)	Static: supplied by Paper Assembler
Fetch Agent	min. = no. of paper-setters	Few seconds to days	1 to no. of setters (10s)	Static initially: supplied by Paper Assembler; Dynamic subsequently
Courier Agent	1 to no. of exam centers	Few minutes	1 to no. of centers (100s)	Static: supplied by Paper Assembler
Question Agent	2 (to – fro student)	1 to 4 hrs (duration of exam.)	No. of students (1000s)	Static: supplied by Distribution Server
Answer Agent	No. of evaluators	1 hr - few months (duration of evaln.)	No. of students (1000s)	Static initially: supplied by Evaluation Server; Dynamic subsequently

5. Mobile Agent advantages in DE

Many of the advantages of MAs, cited in the literature [8] are also applicable to MADE. For example, in MADE almost all the MAs move to the node where the users are situated. By being able to interact locally, users do not become susceptible to varying network delays.

The following additional advantages are specific to MADE:

- *Dynamic content:* MAs carry execution logic along with data and can be used to present dynamic content to the user. In MADE, MAs enable graphical (multimedia) display of data to the student and can carry specific tools such as compilers etc.
- *Hierarchical management:* Control by a central authority and remote management are prime requirements in a distance evaluation application. MAs use generic execution environments. In MADE, we use this property to simplify the infrastructure requirements at different nodes. The managing nodes install most of the application components remotely.
- *Support for both push and pull modes:* MAs can be used to support both the push and pull modes of information dissemination. In MADE, Courier Agents are used to deliver (or push) question papers just-in-time to the students. For examination setting and workflow of answer paper among various evaluators, a combination of both the approaches, viz. push and pull, is used.
- *Force-Fetch:* Partial computations at remote sites may sometimes need to be force-fetched. In MADE, a Question Agent at a student node will time-out after the designated examination duration and return to the server. Similarly, Fetch Agents forcibly bring the partial question papers from the paper-setters after the designated time.
- *Application layer multicasting:* Content carrying MAs can replicate themselves as and when required. This enables application layer multicasting. In MADE, only one copy of the Question Paper is forwarded to each examination center. These get replicated and forwarded to various student nodes.

In the next section we discuss the applicability of MAs to structure large scale distributed applications.

6. MA based structuring of large-scale applications: lessons from MADE

Many proposed MA applications, such as the areas of e-commerce, information retrieval etc. [5] regard MA mainly as a program that performs computations on behalf of the user. We believe that MA approach need not be restricted to this view and that MAs can be extremely useful as an application structuring mechanism.

We feel that MAs may be particularly suitable for structuring large scale distributed applications. We define scale in terms of (i) the number of participating nodes, (ii) the geographical spread of nodes, and (iii) the number of application components.

Based on our experience, we list below some structuring advantages gained by a MA based design.

6.1. Dynamic extensibility

MAs can be used to upgrade an application dynamically. Functionality can be thus added to or removed from the application at run-time. Dynamic extensibility thus helps in automatic software / protocol upgrades at run-time.

In MADE, this property is exploited during the paper-setting stage (see figure 4). The Fetch Agent attaches a GUI object to the paper-setter's application. The paper-setter is then able to manipulate this interface object directly. After the interactions are over, the object is detached from the application.

6.2. Independence from network disconnections

A MA based design can be used to provide support for disconnected operations. Thus dependence on continuous connectivity is reduced. Applications requiring processes to work autonomously for large intervals of time, are good candidates for MA solution.

In MADE, despite complex workflow, continuous connectivity is not required. Message exchanges are required mostly during agent transfers and rarely otherwise. In fact, student terminals can be disconnected from the network for the examination duration.

6.3. Application scalability

A MA based design partitions the application functionality into a number of distributed autonomous units. As a result the addition of more units does not unduly affect the performance of the system.

In MADE, the scalability of the application is required against increase in the number of students, paper-setters, evaluators and examination centers. Addition of these new nodes mainly requires only updating the itineraries of various agents. Since the MAs move to these nodes and use the local resources, no part of the system is overly loaded.

6.4. Ease of restructuring

Any change in application architecture requires components to be added to, relocated or replaced from the system. MA based designs usually result in application components which are autonomous and loosely coupled. MAs themselves can move to various nodes and can be placed where they are best utilized. Hence, restructuring an application may become easier in the case of a MA based design.

In MADE, for example, during the paper-setting process, Install Agents are used to set up the paper-setting infrastructure. A change in system architecture simply involves supplying the new installation rules and components to the Install Agents.

We believe that the above gains in structuring applications arise mainly due to the “Mobile Agent based design” and providing these advantages using traditional client-server paradigms would be exceedingly unwieldy.

We also note that applications that are at present suitable for MA based design are likely to belong to closed environments, i.e., for which all the participating hosts can be trusted. Applications in ‘open environments’ would first need to address additional security concerns.

7. Conclusions

We have built a MA based system for a large-scale distributed application, viz. distance evaluation. We have shown that the Mobile Agent approach is viable and has several advantages for building future Internet applications. Some key advantages of a MA based structuring are: dynamic extensibility, independence from network disconnections, scalability and ease of restructuring.

Our further work is in the following directions:

(i) MADE Mobile Agents vary greatly in life spans, instances and the number of nodes that they visit. There is need for suitable techniques for proper control and management of these different Mobile Agents.

(ii) Managing autonomous mobile components poses many new challenges. Better methods of handling autonomy and improving the overall system reliability need to be studied and formulated.

(iii) Protection of agents (e.g. Answer Agents) from malicious tampering will be a critical requirement when we move from closed to open environments.

8. References

[1] Chien Chou, “Constructing a Computer-Assisted Testing and Evaluation System on the World Wide Web-The CATES Experience”, in *IEEE Transactions on Education*, Vol. 43, No 3, Pages 266-272, August 2000.

[2] Walworth, M., and Herrick, R. J., “The use of computers for educational and testing purposes”, *Proceedings Frontiers in Education Conference,* Pages 510-513, 1991.

[3] Hazari, S. I., “Online Testing Methods for Web Courses”, In *Proceedings of the 14th Annual Conference on Distance Teaching and Learning,* Pages 155-157, November, 1998.

[4] J. White, “Mobile Agents”, in *Software Agents*, J. Bradshaw (ed.), AAAI Press / The MIT Press, 1996.

[5] Alfonso Fuggetta, Gian Pietro Picco and Giovanni Vigna, "Understanding Code Mobility", *IEEE Transactions on Software Engineering* , vol. 24(5), 1998.

[6] Todd Papaioannou, “On Structuring of Distributed Systems, The argument for mobility”, *PhD Thesis*, Loughborough University, 2000.

[7] Neeran Karnik and Anand Tripathi, “Design Issues in Mobile Agent Programming Systems”, *IEEE Concurrency*, Pages 52-61, July-September 1998.

[8] Danny B. Lange, “Mobile Objects and Mobile Agents: The Future of Distributed Computing”, in *Proceedings of The European Conference on Object-Oriented Programming '98*, 1998.

[9] G. Glass, "ObjectSpace Voyager Core Package Technical Overview", *Mobility: process, computers and agents*, Addison-Wesley, February 1999.

[10] Grady Booch, Ivar Jacobson, and James Rumbaugh. The Unified Modeling Language User Guide. *The Addison-Wesley Object Technology Series,* 1999.