DESIGN AND IMPLEMENTATION OF A SOFTWARE AGENT PLATFORM APPLIED IN E-LEARNING AND COURSE MANAGEMENT

A Thesis
Presented to the Institute of Science and Engineering
of
Işık University
In Partial Fulfillment of the Requirements for the Degree of
Master of Science
in
The Department of Computer Engineering

by
Gürol Erdoğan

August 2004

Approval of the Institute of Science and Engineering

_____
Prof. Dr. Sıddık B. Yarman
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____
Prof. Dr. Ahmet Aksen
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____
Prof. Dr. Selahattin Kuru
Supervisor

Examining Commitee Members

.....	_____
.....	_____
.....	_____
.....	_____
.....	_____

ABSTRACT

DESIGN AND IMPLEMENTATION OF A SOFTWARE AGENT PLATFORM APPLIED IN E-LEARNING AND COURSE MANAGEMENT

Erdoğan, Gürol

In this thesis, we report an experience on constructing a software agent platform for development and implementation of software agent systems running with integrated e-learning and course management applications which are developed and running under different technologies. The proposed platform consists of an agent development framework namely JADE (Java Agent Development Environmet), a common database infrastructure serving to many different applications and the applications infrastructure running on different platforms. An example e-university application module which is an integrated course management software running on the proposed platform namely Course ON-LINE and an agent application running as an add-on utility to this application namely GAIA is explained in detail to demonstrate the use of the proposed application.

Keywords: Software Agents, E-University, Course Management, E-Learning

ÖZET

DESIGN AND IMPLEMENTATION OF A SOFTWARE AGENT PLATFORM APPLIED IN E-LEARNING AND COURSE MANAGEMENT

Erdoğan, Gürol

Bu çalışmada farklı teknolojiler kullanılarak geliştirilen ve farklı platformlarda çalıştırılmakta olan ve tümleşik yapıdaki uzaktan eğtim ve ders yönetimi araçları uygulamalarla birlikte çalışabilecek yazılım etmen sistemlerinin geliştirilebilmesini sağlayan bir yazılım geliştirme ve çalıştırma ortamı inşa etme deneyimi aktarılmıştır. Önerilen ortam JADE (Java Agent Development Environmet), isimli bir etmen geliştirme aracı, etmen sistemleri dahil tüm uygulamaların ortak kullandıkları bir veritabanı altyapısı, ve farklı ortamlarda çalışan ve farklı teknolojilerle geliştirilmiş uygulamaların altyapısından oluşmaktadır. Önerilen ortamın kullanılışını göstermek için tümleşik ders web sayfaları yönetim aracı olan ve e-üniversite uygulamalarının bir parçası olan Course ON-LINE ve onunla birlikte çalışan bir yazılım etmeni uygulaması olan GAIA uygulamaları detaylıca sunulmuştur.

Anahtar Kelimeler: Yazılım Etmenleri, E-Üniversite, Ders Web Sayfaları Yönetimi, Uzaktan Eğitim

to my wife

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to all those who contributed directly or indirectly to bringing this publication to this final format, because I would never have been able, by myself, to achieve this.

My most sincere gratitude and appreciation are dedicated to Prof. Dr. Selahattin Kuru, my supervisor, for his inspirational guidance, invaluable suggestions and endless motivation. Many thanks to Mustafa Yıldız, Onur İhsan Arsun, Orhan Karahasan and Ahmet Oktay, my colleagues, for their personal and professional support and for being closest friends.

Finally, I wish to record my special thanks to my parents, for their endless love and confidence.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE TITLE PAGE

LIST OF TABLES

TABLE TITLE PAGE

CHAPTER 1

INTRODUCTION

The popularization of the Internet is generating a great variety of new services for its users, such as e-Commerce, bank transactions, marketing and others. The WWW environment of the Internet has been recognized as a powerful method of information distribution, because it attracts a great number of users and has a low cost. Currently, there is a lot of interest and work in the area of developing agents systems and applications that make use of web technology, which ranges from intelligent information agents, interface agents, to e-commerce agents. Agents are independent software tools linked with other applications and databases running within one or several computer environments. The primary function of an intelligent agent is to help a user better use, manage, and interact with a computer application such as a course management system or a campus portal system [1].

To develop and maintain agent systems running with other web based applications in a particular environment, choosing tools and technologies will be a critical process. This thesis is concerning with building a software agent development and execution platform intended for integrated web based applications and gives detailed information about the technology and tools used in construction of the platform. The products and tools used in different layers and different platforms are explained separately. The proposed platform is intended to be used to develop agent applications running as *add-on* utilities for existing integrated systems sharing common data resources and running on different platforms.

The platform is built on the e-university infrastructure of Işık University and an example agent application is developed which is working as an add-on tool to the *Integrated Course Homepages Management System* of Işık University, namely Course ON-LINE. Course ON-LINE is a member of e-university tools family which includes Campus ON-LINE [2], the course registration and student information

system; Library ON-LINE [3], the university library automation system and CAMPUS ON-SMS [4], information distribution system over SMS. All of these tools are fully integrated with each other and share a common database.

The add-on agent application running within Course ON-LINE is named GAIA. Gaia is a notification agent system, which is running on the Course ON-LINE platform. Students use their own interfaces to start their own notification agents. Student selects the modules of the Course ON-LINE to be tracked by his/her own agent and gives directive to start the agent. This example application shows how to integrate the agent application with the existing applications running on different platforms but using the same data resources.

The second chapter, the agent terminology and a typology of software agents existing in the literature is given. Software agent application domains and benefits of using the software agents are also mentioned in this chapter. This chapter concludes with brief information of publicly and commercially available software agent development tools and software.

Third chapter is dealing with agent applications particularly for educational purposes. This chapter basically explains the properties and basic functionalities of course management and e-learning software and searching for agent applications concerning those products in the literature.

Fourth chapter deeply examines the platform built in the Işık University's e-university infrastructure and explains the tools and technologies used in the integrated platform using software agents.

Fifth chapter is about the example application Course ON-LINE and its agent add-on namely GAIA. The core functionalities of these products are presented in this chapter. This chapter also includes the programming methodology and software patterns used in development of Course ON-LINE and GAIA.

CHAPTER 2

SOFTWARE AGENT SYSTEMS

2.1. Software Agent Definitions

People involved in agent research have covered a variety of definitions on software agents, each hoping to explicate his or her use of the word, *'agent'*. These definitions range from the simple to the lengthy and demanding. Each of them grew directly out of the set of examples of agents that the definer had in mind. By an *agent*, we mean a system that has the following properties [5]:

– *autonomy*: agents encapsulate some state (that is not accessible to other agents), and make decisions about what to do based on this state, without the direct intervention of humans or others;

– *reactivity*: agents are *situated* in an environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps many of these combined), are able to *perceive* this environment (throughthe use of potentially imperfect sensors), and are able to respond in a timely fashionto changes that occur in it;

– *pro-activeness*: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by *taking the initiative*;

– *social ability*: agents interact with other agents (and possibly humans) via some kind of *agent-communication language* [28], and typically have the ability to engage in social activities (such as cooperative problem solving or negotiation) in order to achieve their goals.

These properties are more demanding than they might at first appear. To see why, let us consider them in turn. First, consider *pro-activeness*: goal directed behavior. It is not hard to build a system that exhibits goal directed behavior — we do it every time we write a procedure in Pascal, a function in C, or a method in Java. When we write such a procedure, we describe it in terms of the *assumptions* on which it relies (formally, its *pre-condition*) and the *effect* it has if the assumptions are valid (its *post-condition*). The effects of the procedure are its *goal*: what the author of the software intends the procedure to achieve. If the pre-condition holds when the procedure is invoked, then we expect that the procedure will execute *correctly*: that it will terminate, and that upon termination, the post-condition will be true, i.e., the goal will be achieved. This is goal directed behavior: the procedure is simply a plan or recipe for achieving the goal. This programming model is fine for many environments. For example, its works well when we consider *functional systems* — those that simply take some input, and produce as output some some function of this input. Compilers are a classic example of functional systems.

But for non-functional systems, this simple model of goal directed programming is not acceptable, as it makes an important limiting assumption. It assumes that the environment *does not change* while the procedure is executing. If the environment does change, and in particular, if the assumptions (pre-condition) underlying the procedure become false while the procedure is executing, then the behavior of the procedure may not be defined — often, it will simply crash. Similarly, it is assumed that the goal, that is, the reason for executing the procedure, remains valid at least until the procedure terminates. If the goal does *not* remain valid, then there is simply no reason to continue executing the procedure.

In many environments, neither of these assumptions are valid. In particular, in domains that are *too complex* for an agent to observe completely, that are *multi-agent* (i.e., they are populated with more than one agent that can change the environment), or where there is *uncertainty* in the environment, these assumptions are not reasonable. In such environments, blindly executing a procedure without regard to whether the assumptions underpinning the procedure are valid is a poor strategy. In such dynamic environments, an agent must be *reactive*, in just the way that we described above. That is, it must be responsive to events that occur in its

environment, where these events affect either the agent's goals or the assumptions which underpin the procedures that the agent is executing in order to achieve its goals.

As we have seen, building purely goal directed systems is not hard. Similarly, building *purely reactive* systems — ones that *continually* respond to their environment— is also not difficult; we can implement them as lookup tables that simply match environmental stimuli to action responses. However, what turns out to be very hard is building a system that achieves an effective *balance* between goal-directed and reactive behavior. We want agents that will attempt to achieve their goals systematically, perhaps by making use of complex procedure-like recipes for action. But we don't want our agents to continue blindly executing these procedures in an attempt to achieve a goal either when it is clear that the procedure will not work, or when the goal is for some reason no longer valid. In such circumstances, we want our agent to be able to react to the new situation, in time for the reaction to be of some use. However, we do not want our agent to be *continually* reacting, and hence never focusing on a goal long enough to actually achieve it.

On reflection, it should come as little surprise that achieving a good balance between goal directed and reactive behavior is hard. After all, it is comparatively rare to find humans that do this very well. How many of us have had a manager who stayed blindly focussed on some project long after the relevance of the project was passed, or it was clear that the project plan was doomed to failure? Similarly, how many have encountered managers who seem unable to stay focussed at all, who flit from one project to another without ever managing to pursue a goal long enough to achieve *anything*? This problem—of effectively integrating goal-directed and reactive behavior—is one of the key problems facing the agent designer. As we shall see, a great many proposals have been made for how to build agents that can do this—but the problem is essentially still open.

Finally, let us say something about *social ability*, the final component of flexible autonomous action as defined here. In one sense, social ability is trivial: every day, millions of computers across the world routinely exchange information with both humans and other computers. But the ability to exchange bit streams is not

really social ability. Consider that in the human world, comparatively few of our meaningful goals can be achieved without the *cooperation* of other people, who cannot be assumed to *share* our goals — in other words, they are themselves autonomous, with their own agenda to pursue. This type of social ability—involving the ability to dynamically negotiate and coordinate — is much more complex, and much less well understood, than simply the ability to exchange bitstreams.

An obvious question to ask is why agents and multi-agent systems are seen as an important new direction in software engineering. There are several reasons [40, pp.6– 10]:

– *Natural metaphor*.

Just as the many domains can be conceived of consisting of a number of interacting but essentially passive *objects*, so many others can be conceived as interacting, active, purposeful *agents*. For example, a scenario currently driving much R&D activity in the agents field is that of software agents that buy and sell goods via the Internet on behalf of some users. It is natural to view the software participants in such transactions as (semi-)autonomous agents.

– *Distribution of data or control*.

For many software systems, it is not possible to identify a single locus of control: instead, overall control of the systems is distributed across a number computing nodes, which are frequently geographically distributed. In order to make such systems work effectively, these nodes must be capable of autonomously interacting with each other— they must agents.

– *Legacy systems*.

A natural way of incorporating legacy systems into modern distributed information systems is to *agentify* them: to “wrap” them with an agent layer, that will enable them to interact with other agents.

– *Open systems*.

Many systems are *open* in the sense that it is impossible to know at design time exactly what components the system will be comprised of, and how these

components will be used to interact with one-another. To operate effectively in such systems, the ability to engage in flexible autonomous decision-making is critical.

2.2. A Typology of Software Agents

This section places existing agents into different agent classes. A typology refers to the study of types of entities. There are several dimensions to classify existing software agents. [6]

Firstly, agents may be classified by their mobility, i.e. by their ability to move around some network. This yields the classes of static or mobile agents.

Secondly, they may be classed as either deliberative or reactive. Deliberative agents derive from the deliberative thinking paradigm: the agents possess an internal symbolic, reasoning model and they engage in planning and negotiation in order to achieve coordination with other agents. Reactive agents on the contrary do not have any internal, symbolic models of their environment, and they act using a stimulus/response type of behavior by responding to the present state of the environment in which they are embedded.

Thirdly, agents may be classified along several ideal and primary attributes which agents should exhibit. At BT Labs, they have identified a minimal list of three: autonomy, learning and cooperation. Autonomy refers to the principle that agents can operate on their own without the need for human guidance, even though this would sometimes be invaluable. Hence agents have individual internal states and goals, and they act in such a manner as to meet its goals on behalf of its user. A key element of their autonomy is their proactiveness, i.e. their ability to ’take the initiative’ rather than acting simply in response to their environment. Cooperation with other agents is paramount: it is the reason for having multiple agents in the first place in contrast to having just one. In order to cooperate, agents need to possess a social ability, i.e. the ability to interact with other agents and possibly humans via some communication language. Having said this, it is possible for agents to coordinate their actions without cooperation. Lastly, for agent systems to be truly ’smart’, they would have to

learn as they react and/or interact with their external environment. The learning may also take the form of increased performance over time. They use these three minimal characteristics in Figure 2.1. to derive four types of agents to include in their typology: collaborative agents, collaborative learning agents, interface agents and truly smart agents. [5]

Fourthly, agents may sometimes be classified by their roles (preferably, if the roles are major ones), e.g. World Wide Web (WWW) information agents. This category of agents usually exploits internet search engines such as WebCrawler, Lycos and Spiders. Essentially, they help manage the vast amount of information in wide area networks like the Internet. They refer to these classes of agents as information or internet agents. Again, information agents may be static, mobile or deliberative.

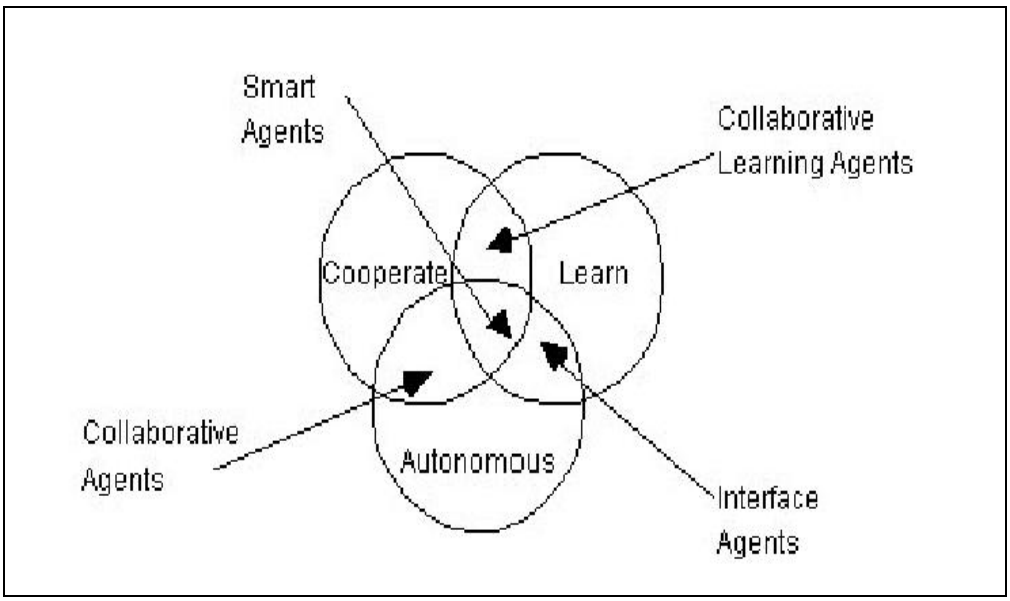

Figure 2.1 A Part View of an Agent Typology

Fifthly, they have also included the category of hybrid agents that combine of two or more agent philosophies in a single agent. There are other attributes of agents that they consider secondary to those already mentioned. For example, is an agent versatile (i.e. does it have many goals or does it engage in a variety of tasks)? Is an agent benevolent or non-helpful, antagonistic or altruistic? Does an agent lie knowingly or is it always truthful (this attribute is termed veracity)? Can you trust

the agent enough to (risk) delegate tasks to it? Is it temporally continuous? Does it degrade gracefully in contrast to failing drastically at the boundaries? Perhaps unbelievably, some researchers are also attributing emotional attitudes to agents - do they get 'fed up' being asked to do the same thing time and time again? What role does emotion have in constructing believable agents (Bates, 1994)? Some agents are also imbued with mentalist attitudes or notions such as beliefs, desires and intentions - referred to typically as BDI agents.

In essence, agents exist in a truly multi-dimensional space. For the sake of clarity of understanding, they have 'collapsed' this multi-dimensional space into a single list. In order to carry out such an audacious move, they have made use of their knowledge of the agents they know are currently 'out there'. Therefore, the ensuing list is to some degree arbitrary, but these types cover most of the agent types being investigated currently. There have been identified seven types of agents:

- Collaborative agents
- Interface agents
- Mobile agents
- Information/Internet agents
- Reactive agents
- Hybrid agents
- Smart Agents

There are some applications that combine agents from two or more of these categories, and we refer to these as heterogeneous agent systems. Such applications already exist even though they are relatively few. The next sections give a brief description of each type of agent.

2.2.1. Collaborative Agents

As shown in Figure 2.1 collaborative agents emphasize autonomy and cooperation (with other agents) in order to perform tasks for their owners. They may

learn, but this aspect is not typically a major emphasis of their operation. In order to have a coordinated set up of collaborative agents, they may have to negotiate in order to reach mutually acceptable agreements on some matters.

2.2.2. Interface Agents

Interface agents emphasize autonomy and learning in order to perform tasks for their owners.

The key metaphor underlying interface agents is that of a personal assistant who is collaborating with the user in the same work environment. Note the subtle emphasis and distinction between collaborating with the user and collaborating with other agents as is the case with collaborative agents. Collaborating with a user may not require an explicit agent communication language as one required when collaborating with other agents.

2.2.3. Mobile Agents

Mobile agents are computational software processes capable of roaming wide area networks (WANs) such as the WWW, interacting with foreign hosts, gathering information on behalf of its owner and coming ’back home’ having performed the duties set by its user. These duties may range from a flight reservation to managing a telecommunications network.

2.2.4. Information/Internet Agents

Information agents have come about because of the sheer demand for tools to help us manage the explosive growth of information we are experiencing currently, and which we will continue to experience henceforth. Information agents perform the role of managing, manipulating or collating information from many distributed sources. Information or Internet agents are defined by what they do, in contrast to collaborative or interface agents which we defined by what they are.

2.2.5. Reactive Agents

Reactive agents represent a special category of agents which do not possess internal, symbolic models of their environments; instead they act/respond in a stimulus-response manner to the present state of the environment in which they are embedded. Three key ideas which underpin reactive agents: firstly, there is no a priori specification (or plan) of the behavior of the set-up of reactive agents. Secondly, is that of 'task decomposition': a reactive agent is viewed as a collection of modules that operate autonomously and are responsible for specific tasks (e.g. sensing, motor control, computations, etc.). Communication between the modules is minimized and of quite a low-level nature. Thirdly, reactive agents tend to operate on representations that are close to raw sensor data, in contrast to the high-level symbolic representations that abound in the other types of agents discussed so far.

2.2.6. Hybrid Agents

Hybrid agents refer to those whose constitution is a combination of two or more agent philosophies within a singular agent. Since each type has (or promises) its own strengths and deficiencies, the trick (as always) is to maximize the strengths and minimize the deficiencies of the most relevant technique for your particular purpose.

2.2.7. Heterogeneous Agent Systems

Heterogeneous agent systems, unlike hybrid systems described in the preceding section, refer to an integrated set-up of at least two or more agents that belong to two or more different agent classes. A heterogeneous agent system may also contain one or more hybrid agents.

2.3. FIPA Agent Specification

The Foundation for Intelligent Physical Agents (FIPA) [7] is a multi-disciplinary group pursuing the standardization of agent technology. This organization has made available a series of specifications to direct the development of multi-agent systems. Of particular importance are their Agent Management and Agent Communication Language specifications. FIPA's approach to MAS development is based on a "minimal framework for the management of agents in an open environment." This framework is described using a reference model (which specifies the normative environment within which agents exist and operate), and an agent platform (which specifies an infrastructure for the deployment and interaction of agents).

The FIPA architecture defines at an abstract level how two agents can locate and communicate with each other by registering them and exchanging messages. To do this, a set of architectural elements and their relationships are described. In this section the basic relationships between the elements of the FIPA agent system are described. This section gives a relatively high level description of the notions of the architecture. It does not explain all of the aspects of the architecture.

2.3.1. Agents and Services

Agents communicate by exchanging messages which represent speech acts, and which are encoded in an agent-communication-language.

Services provide support services for agents. This version of the Abstract architecture defines two support services: directory-services and message-transport-services.

Services may be implemented either as agents or as software that is accessed via method invocation, using programming interfaces such as those provided in Java, C++, or IDL. An agent providing a service is more constrained in its behavior than a general-purpose agent. In particular, these agents are required to preserve the semantics of the service. This implies that these agents do not have the degree of

autonomy normally attributed to agents. They may not arbitrarily refuse to provide the service.

2.3.2. Agent Communication Languages

Once we have achieved a way of representing the knowledge of our agents, we need tools for sharing and exchanging that knowledge. There are two main initiatives to this end.

The Foundation for Intelligent Physical Agents (FIPA) has a proposed an Agent Communication Language (ACL) that is founded on Speech Act Theory. The FIPA-ACL abstracts away low level communication details and assumes the existence of an Agent Management System not part of the languages. [7]

ACL consists of an inner context language with a common vocabulary and ontology, and an outer communication language, and message passing mechanisms. Inner context languages are usually KIF, Eclipse Prolog, Tcl/Tk, Java, or the Java Agent Template. The outer communication language is almost always KQML.

The Knowledge Query and Manipulation Language (KQML) is perhaps the most widely used communication formalism. KQML was developed as a part of the Knowledge Sharing Effort. [9]

KQML is built around a number of performatives designed to achieve tasks at three conceptual layers: Content - Message - Communication. The attempt in KQML to be able to handle tasks at several levels of abstractions is sometimes put forward as a significant drawback of the language. Distinctions between layers are not directly evident in programs and specifications, something that is confusing and make building good abstractions difficult.

2.4. Benefits of Agent Applications

Distributed systems based on the concepts of agents and places, of agent migration and agent communication simplifies the implementation of many applications. At the same time they make new kinds of applications with novel functionalities possible. In the following we discuss a sampling of these uses. As has been pointed out by Chess et al. [12], most of the benefits of mobile agents could also be achieved by using other means, but mobile agents offer all these benefits in a single framework.

2.4.1. Reduction of Communication

Although a certain overhead for sending agent code and execution state across the network must be considered, mobile agents can reduce communication with respect to latency, bandwidth and connection time. Communication latency can be reduced by sending an agent with a sequence of service requests across the network rather than issuing each service request by a separate remote procedure call. Communication bandwidth can be reduced by moving the agent across the network in order to deliver instructions for the generation of data on a remote host. A performance model for communication in mobile agent systems has been given by Strasser and Schwehm [13]. An example for the reduction of communication by mobile code is the NeWS window system. In NeWS, clients communicate with the display server by sending PostScript programs. Instead of drawing a grid by sending several thousand messages for individual points, it is possible to send one brief program that will compute and draw the entire grid. The code sent by the client can also be used to extend the server, so that complex actions can be carried out in the future using a single message. Communication bandwidth can also be reduced by moving the agent across the network to the source of data in order to reduce the data before transmission. For example, an information gathering agent can roam the network, where it queries several remote databases and filters the results in order to return only the best 10 matches. The reduction of connection time is important in the context of mobile computing. The information gathering agent from the previous example could be uploaded from a mobile computer. The mobile computer need only

be connected to the network while uploading the agent, and eventually to a later time to gather the agent's results.

2.4.2. Asynchronous Tasks

Asynchronous communication mechanisms, such as asynchronous message queues [14] allow for asynchronous processing of requests. While the individual requests of a task can be processed asynchronously, the client performing this task must be available to receive and react on incoming replies. Keeping a mobile client up and connected while task processing is in progress might be expensive at least or even impossible. With agent technology, the client part of the application can be transferred from the mobile device to stationary servers in the network. From an end user's perspective, not only individual requests but the entire task is moved to the network, where it is performed asynchronously. Clearly, once the task transfer is complete, the mobile device can be disconnected from the network. Later, after hours or even days, the device can be reconnected to receive the task's results. It is important to notice that the underlying assumption of those scenarios is that the underlying system guarantees 'exactly once' semantics of agents, i.e. when accepting an agent, the network guarantees that the agent is not lost and is performed exactly once, independent of communication and node failures. Unfortunately, none of the current agent systems supports this level of fault-tolerance.

2.4.3. Dynamic Protocols and Intelligent Data

The rapid growth of the internet has also increased the number of protocols and data formats for data exchange between computers. A computer generally supports only a limited number of protocols. If a particular protocol is missing in order to access, view or process some received data, the protocol must be installed manually. Mobile agents permit dynamic protocols, i.e. new protocols to be installed automatically and only as needed for a particular interaction. To receive an agent initially, the client and server must share some standard protocol. Once the agent is running, though, it can use a specialized protocol for communication back to its

home server. Furthermore, an executing agent can communicate repeatedly with the server without intervention from the user, allowing the construction of dynamic services. For example a news service could transmit news updates to agents on distributed clients by using a special multicast protocol. Associating agents with data provides a way for the data to know how to process itself. A recent example of intelligent data is the MPEG4 compression standard for video, where the decompression algorithm is bundled with the data. This approach makes the standard highly flexible and allows the upgrade to use improved compression techniques.

2.4.4. Software Deployment

Mobile agents can be used to automate the software installation and updating process. Next to the transport of the software package, the agent can gather information about the environment, query the user for installation preferences, configure the system, create directories and uncompress and compile the software. After successful installation, the agent can become responsible to gather software updates. This approach to software deployment has its limitations since it might not be possible to capture every special case and error condition of the installation process and the programming of suitable deployment agents might become very difficult. Furthermore, it has to be considered that such a software deployment might only be applicable within a trusted environment (LAN), since software coming from an untrusted source could (purposely or by accident) delete or damage the data on its new host. A better approach to software deployment would be to use the agent language itself, since the agent language is in particular designed to prevent such damage.

2.4.5. Temporary Applications

A mobile agent is not limited to deploy software packages, the agent could be the application itself. An application-agent might be self-contained and have no communication or migratory needs at all. The application agent would be much smaller than a stand-alone application since it could exploit the infrastructure

provided by the mobile agent system. After downloading the application agent, no creating of directories, configuring, compiling and installing would be necessary. This simplicity allows to download applications temporarily and to discard them after usage. Examples of temporary applications can be travel guides and route planners downloaded on a mobile computer for a particular trip and discarded afterwards. Upon arrival at a new location, the user might temporarily download services that are specific to the new environment. Very popular examples of such application agents are applets written in the Java language. Applets are self-contained programs contained in Web pages that start execution once they have been downloaded.

2.4.6. Distributed and Heterogeneous Computing

Mobile agents can also serve as the basis for general-purpose distributed and heterogeneous computing. According to the needs for a distributed program, the agents migrate to their computer node and execute their scheduled task. The mobile agent system provides the necessary infrastructure for communication between the tasks in a heterogeneous environment. The agent system furthermore supports the independent compilation and initiation of agents so that further agents can be assigned to a task at runtime. Furthermore many algorithms can more naturally be expressed in terms of mobility through a network rather than message passing. Prospective applications for agent-based distributed computing are parallel algorithms with a reasonable low communication overhead compared to its computation requirements and particle or object based simulations. Note that agent-based distributed computing does not necessarily require agent mobility.

Adding value to a service - e.g. plotting a graph instead of submitting raw data - poses a problem to highly frequented servers, since the value added server reduces the anyway limited computational resources. Submitting the raw data together with a corresponding agent for graph plotting would move the computation to the client. This has two advantages: Transfer of raw data and agent might be cheaper than transferring an image and the computation is moved from the server to the client which might accept this since he is waiting for the graph anyway. Furthermore, the

agent might provide an interface to customize the graph according to the needs of the client. On the other hand, if the client uses a small mobile device and the raw data should be used for a complex image rendering, the user might not want the agent to execute on the mobile computer but on an intermediate compute-server. Mobile agents provide an environment which allows moving computation to nodes with appropriate resources, thus load-balancing a distributed system.

2.4.7. Scalable Applications

Dynamic deployment of agent programs allow for more scalable applications. Assume, for example, a search application that accesses a large number of globally distributed data sources. Assume documents are retrieved from the data sources and selected (or indexed) based on a content-based filtering function. In a pure client/server setting, a client would access the remote data sources, and all retrieved documents would be transferred to the client. The final filtering would be performed at the client site. If accessing the data sources is performed in parallel, the client as well as (parts of) the network may become a bottleneck. With mobile agents, a hierarchy of filter agents can be set up. Filter agents not only perform content-based filtering but also get rid of redundantly retrieved documents. The structure of the hierarchy and the placement of the individual filter agents mainly depend on the set of data sources accessed. Both placement and structure can change if new data sources are detected while the search operation is in progress. Obviously, this setting is more scalable since filtering is distributed and can be performed close at the data sources. Moreover, redundant information can be detected early and thus must not travel all the way to the client.

2.5. Agent Development Tools

Whilst programming languages like Java and C++ provide an extensive library of classes for general purpose software development, such libraries tend not to include high-level constructs and concepts needed for agent applications. Consequently, there is an emerging consensus amongst agent researchers of the need

to develop methodologies and tool-kits for building distributed agent systems. This equates to moving away from point solutions and towards general architectures, frameworks and tool-kits.

2.5.1. BT's ZEUS

According to ZEUS, each agent consists of a definition layer, an organizational layer and a co-ordination layer. The Definition Layer comprises the agent's reasoning (and learning) abilities, its goals, resources, skills, beliefs, preferences, etc. The organization layer describes the agent's relationships with other agents, e.g. what agencies it belongs to, what abilities it knows other agents possess, etc. At the co-ordination layer the agent is modeled as a social entity, i.e. in terms of the co-ordination and negotiation techniques it possesses. Built on top of the co-ordination layer are the communication protocols that implement inter-agent communication; whilst beneath the definition layer is the application programmer's interface (API) that links the agent to the physical realizations of its resources and skills.

2.5.2. Grasshopper

Grasshopper, which has been developed by GMD FOKUS and IKV++ GmbH, is a mobile agent development and runtime platform which is built on top of a distributed processing environment. This achieves an integration of the traditional client/server paradigm and mobile agent technology. Grasshopper is implemented in Java, based on the Java 2 specification. Most importantly, Grasshopper has been designed in conformance with the first mobile agent industry standard, namely the Object Management Group's Mobile Agent System Interoperability Facility (MASIF). In addition, the latest Grasshopper version is also compliant with the specifications of the Foundation for Intelligent Physical Agents (FIPA).

2.5.3. Concordia

Concordia is developed at Mitsubishi Electric ITA Horizon Systems Laboratory. It is a full-featured framework for the development and management of network client mobile agent applications which extend to any device supporting Java. Concordia is written in Java and is portable to any platform running Java. A Concordia System, at its simplest, is made up of a Java Virtual Machine (VM), a Concordia Server, and at least one mobile agent on 1 network node. Usually, the Concordia System will consist of multiple machines in a local or wide area network, each of which is running Java Virtual Machines, Concordia Servers, and mobile agents.

2.5.4 IBM Aglets

The Aglets Software Development Kit is an environment for programming mobile Internet agents in Java. It is what was used to be called Aglets Workbench.

The aglet represents the next leap forward in the evolution of executable content on the Internet, introducing program code that can be transported along with state information. Aglets are Java objects that can move from one host on the Internet to another. That is, an aglet that executes on one host can suddenly halt execution, dispatch itself to a remote host, and resume execution there. When the aglet moves, it takes along its program code as well as its data.

CHAPTER 3

AGENT APPLICATIONS IN E-LEARNING AND COURSE MANAGEMENT

3.1. E-Learning and Course Management

The Internet is being widely used in education already, as a source of information it is almost unrivalled for speed and accessibility to a huge range of data. This is one of the important reasons why the Internet became a good reference tool and also a teaching aid. With almost instantaneous communication anywhere in the world, information, be that lecture notes or shared projects, can be passed between students and teachers with reliability and ease. [15]

With the introduction of distance learning and virtual university concepts, the academic world began to argue which one is the best way of teaching. Mostly accepted idea is that existing universities must assimilate new communications technologies and new ways of teaching, learning, research and communications are necessary in order to prevent extinction. In other words, the traditional universities are going to be virtualized rather than to be replaced. [16]

Currently many instructors use Internet for creating course homepages for their own lectures and share their resources over Internet. They communicate with their students mostly using e-mail.

A good way to encourage the use of these new technology tools in universities is to supply a uniform framework and form an integrated course homepage management system.

Integrated course homepage management has the following advantages:

- Standardized interfaces for every course homepage. This makes it easy for students and instructors to learn and use the system and gives opportunity for every instructor to use same number of functionality on their own course homepages.
- Centralized management of the whole electronic resources of the university gives instructors the opportunity to manage their electronic content easer and more controlled.
- Availability of course contents as a university asset with uniform and managed course content
- Provides a tool for school administration to promote widespread use of web resources and technology in education.

The availability of a uniform and managed course materials is an important asset for a university to provide access to course materials for all educators, students, and self-learners around the world to share the research, pedagogy, and knowledge to benefit others which is an important mission of a university.

Web support for courses is becoming increasingly important. An interesting example is that MIT (Massachusetts Institute of Technology) announced a project named Open Courseware [17] which aims to put all electronic content of MIT to the web and make them publicly accessible. In the beginning of 2004, over 500 courses' contents are being published in this project.

3.2. Functions of E-Learning and Course Management Tools

This section beriefly lists expected functions of electronic learning and course management systems. The functionalities are categorized in the hierarcy given in Figure 3.1. [18]

- Learner Tools
  - Communication Tools
    - Discussion Forums
    - File Exchange
    - Internal Email
    - Online Journal/Notes
    - Real-time Chat
    - Video Services
    - Whiteboard
  - Productivity Tools
    - Bookmarks
    - Calendar/Progress Review
    - Orientation/Help
    - Searching Within Course
    - Work Offline/Synchronize
  - Student Involvement Tools
    - Groupwork
    - Self-assessment
    - Student Community Building
    - Student Portfolios
- Support Tools
  - Administration Tools
    - Authentication
    - Course Authorization
    - Hosted Services
    - Registration Integration
  - Course Delivery Tools
    - Automated Testing and Scoring
    - Course Management
    - Instructor Helpdesk
    - Online Grading Tools
    - Student Tracking

- Curriculum Design
  - Accessibility Compliance
  - Content Sharing/Reuse
  - Course Templates
  - Curriculum Management
  - Customized Look and Feel
  - Instructional Design Tools
  - Instructional Standards Compliance

Figure 3.1 CMT Functionalities Hierarchy

Following sections explain each functionality in more detail.

3.2.1. Learner Tools

3.2.1.1. Communication Tools

Discussion Forums

Discussion forums are online tools that capture the exchange of messages over time, sometimes over a period of days, weeks, or even months. Threaded discussion forums are organized into categories so that the exchange of messages and responses are grouped together and are easy to find.

Discussion forums tools are very similar to Usenet newsgroups where text conversations over time are displayed. The organization of the messages can be a simple temporal sequence or they can be presented as a threaded discussion where only messages on a specific topic called a thread are displayed in sequence.

File Exchange

File exchange tools allow learners to upload files from their local computers and share these files with instructors or other students in an online course. Note: File attachments to messages are part of Internal Email and Discussion Forums.

File Exchange tools enable downloading files and upload or posting files over the Web from within the course (a.k.a. assignment drop box).

Internal Email

Internal email is electronic mail that can be read or sent from inside an online course.

Email tools enable messages be read and sent exclusively inside the course or alternatively the tools enable links to external email addresses of those in the course so that contacting course members is facilitated. Internal email may include an address book and some address books are searchable.

Online Journal/Notes

Online Notes/Journal enable students to make notes in a personal or private journal. Students can share personal journal entries with their instructor or other students but cannot share private journal entries.

Online Note/Journal tool enables students to make notes about course experiences. These notes can be personal or private. Students can share personal notes with an instructor or other students. They cannot share private journal entries. This tool can be used to facilitate writing assignments where parts are written over time and then later assembled into a document. This tool also can be used to make personal annotations to pages of a course that can later be used as a study aide. The Online Notes tool can also be used to record reflections about personal learning accomplishments and how to apply this new knowledge.

Real-time Chat

Real-time chat is a conversation between people over the Internet that involves exchanging messages back and forth at virtually the same time.

Chat includes facilities like Internet Relay Chat (IRC), instant messaging, and similar text exchanges in real time. Some chat facilities allow the chats to be archived for later reference. Some chats can be moderated, similar to the notion of

"passing the microphone." Other chats can be monitored, where an instructor can view the conversation in a room without their presence being broadcast.

Video Services

Video services enable instructors to either stream video from within the system, or else enable video conferencing, either between instructors and students or between students.

Video Services include tools for broadcasting video to those without a video input device. Some video services provide for two-way or multi-way video conferencing which may be point-to-point connections or mediated through a central server. See also Real-Time Chat and Whiteboard.

Whiteboard

Whiteboard tools include an electronic version of a dry-erase board used by instructors and learners in a virtual classroom (also called a smartboard or electronic whiteboard) and other synchronous services such as application sharing, group browsing, and voice chat.

Application sharing allows a software program running on one computer to be viewed, and sometimes controlled from a remote computer. For example, an instructor using this feature can demonstrate a chemistry experiment or a software utility to an online student and allow the student to use the demonstration software from their own computer. Group Web Browsing allows an instructor to guide learners on a tour of web sites using a shared browser window. Voice chat allows two or more to communicate in real time via microphones, conference call style, over an Internet connection.

3.2.1.2. Productivity Tools

Bookmarks

Bookmarks allow students to easily return to important pages within their course or outside their course on the web. In some cases bookmarks are for an

individual students private use, and in others can be shared with an instructor or amongst an entire class. Some systems also allow bookmarks to be annotated.

Bookmarks allow students to easily return to important pages within their course or outside their course on the web. Systems vary in allowing students to store their bookmarks in a course folder, a personal folder, or a private folder. Course folders are open to all students and instructors in a course. Personal folders contain bookmarks that individual students can share whereas bookmarks in private folders are for the students own use. Bookmarks can sometimes be annotated and categorized within folders.

Calendar/Progress Review

Calendar/Progress Review tools enable students to document their plans for a course and the associated assignments in a course.

Calendar/Progress Review tools often enable students to check their marks on assignments and test, as well as their progress through the course material. Students can sometimes compare their marks on an assignment with the average score on that assignment, view total points earned, total points possible and percentages per unit, per item and overall course grade.

Orientation/Help

Orientation/Help tools are designed to help students learn how to use the course management system. Typically, these tools are self-paced tutorials, user manuals, and email or telephone helpdesk support.

Orientation/Help tools enable students to make the best use of the software. These tools provide instruction about and job aids for using various aspects of the course management system. Student support tools may include context sensitive help, hints, and wizard style assistants. Some product providers include courses in how to study effectively and/or how to work in online groups. A student helpdesk does not typically offer help with course content.

Searching Within Course

Searching within a course is a tool that allows users to find course material based on key words.

Searching tools enable students to locate parts of the course materials on the basis of word matching beyond the user's current browser page (which can be searched using the browser>edit>find menu).

Work Offline/Synchronize

Work offline/synchronize is a set of tools that enable students to work offline in their online course and for their work to be synchronized into the course the next time they log-in. Sometimes students download course content to their local computers and sometimes they access content on a CD-ROM. Course content that resides on a CD-ROM can also be linked to dynamically within the online course. A course placeholder automatically returns students to the location in their course where they were working the last time they logged off.

The ability to work in a course environment offline and/or to automatically return to the location in the course where you were working the last time you logged off, is especially useful in situations where communication links are unreliable or expensive. The offline environment is essentially a local client application that embodies the important features of the online product without a continuous connection to the Internet. Tracking and student performance data are automatically uploaded into and synchronized with the student performance database the next time the student logs in. The course placeholder tool is essentially an automated bookmark that returns users directly to the page of the course where they had stopped working the last time they logged off.

3.2.1.3. Student Involvement Tools

Groupwork

Group Work is the capacity to organize a class into groups and provide group work space that enables the instructor to assign specific tasks or projects.

Some systems also enable groups to have their own communications features like real-time chat and discussion forums.

Self-assessment

Self-assessment tools allow students to take practice or review tests online. These assessments do not count toward a grade.

Self assessments encourage students to take responsibility for their own learning and to monitor their learning progress. Self assessments can also facilitate student motivation if students receive feedback on the self-assessments and if there is a direct connection between the self assessments and the measurement instruments the instructor uses to determine final course grades. Note: For information on the different question formats, e.g., multiple choice or fill-in-the-blank, see Automated Testing and Scoring.

Student Community Building

Student Community Building tools allow students to create study groups, clubs, or collaborative teams.

Student Community Building tools can encourage and support the growth of student friendships and partnerships. Some products enable students to create and manage these groups. Some products also allow these groups to be formed at the system level, rather than the course level. See also Discussion Forums, File Exchange, Real-Time Chat, and Groupwork.

Student Portfolios

Student Portfolios are areas where students can showcase their work in a course, display their personal photo, and list demographic information.

Student Portfolios are often located on or are a part of students personal homepages in each course. Some products provide a private folder and a public course or team folder that students can use to display their work. Students personal homepages typically give them access to course content, internal email, course announcements, and the course calendar. See also Calendar/Progress Review for tools that allow students to track their progress in a course.

3.2.2. Support Tools

3.2.2.1. Administration Tools

Authentication

Authentication is a procedure that works like a lock and key by providing access to software by a user who enters the appropriate user name (login) and password. Authentication also refers to the procedure by which user names and passwords are created and maintained.

Authentication systems can involve a single logon which is the most user friendly and most vulnerable to hacking. More complicated systems can involve layers with separate logins for each layer and secure socket layer transaction (SSL)encryption.

Course Authorization

Course authorization tools are used to assign specific access privileges to course content and tools based on specific user roles, e.g. students, instructors, teaching assistants. For example, students can view pages and instructors can author pages.

Students and instructors typically need different tools to complete their instructional responsibilities. For example, students need to be able to view their records in a grade book but instructors need to be able to view and modify the records of all students in the course. Most course management systems provide a small set of default user roles. Some systems allow institutions to add and define additional user roles.

Hosted Services

Hosted Services means that the product provider offers the course management system on a server at their location so the institution does not provide any hardware.

An important aspect of Hosted Services is that the product provider takes responsibility for all technical support and maintenance of the server, as well as the actual web service of providing online courses.

Registration Integration

Registration tools are used to add students to and drop students from an online course. Administrators and/or instructors use registration tools but students also use them when self-registration is available. Students can also be added to or dropped from an online course through integration of the course management system with a Student Information System(SIS). Registration tools include secure credit card transactions.

Some registration tools allow administrators or instructors to add or drop students in batches through the use of formatted text files. Time limited student self-registration may also be available to shift the clerical burden of the process to the students. Registration tools include the integration of the course management system with an administrative student registration or information system. Integration with Student Information Systems (SIS) enables the course management system to work with products such as SCT Banner, Peoplesoft, or Datatel. Typically, integration allows the following types of functionality: shared common student information, ability to transfer grades between the SIS and the course management system, and the ability to have common accounts. The registration tools for secure transactions involve making arrangements with financial institutions for the funds to be transferred to the college or university. These arrangements may have a separate cost structure. See also Authentication for information on secure socket layer transaction (SSL) encryption. See also Optional Extras for third party credit card support and international pricing.

3.2.2.2. Course Delivery Tools

Automated Testing and Scoring

Automated Testing and Scoring tools allow instructors to create, administer, and score objective tests.

Some products provide support for proctored testing in a suitable computer lab classroom as an approach to ensuring academic honesty. Note: See also Online Grading, Self Assessment, and Student Tracking.

Course Management

Course management tools allow instructors to control the progression of an online class through the course material.

Course Management tools are used to make specific resources in a course, such as readings, tests or discussions, available to students for a limited time only or after some prerequisite is achieved. This deliberate unfolding of the course resources can be used to prevent students from being overwhelmed and discouraged. Some systems enable this course management to be individualized so that course experience can be tailored to accommodate individual learner situations. Note: The management of testing is covered in the Automated Testing and Scoring feature.

Instructor Helpdesk

Instructor Helpdesk tools help faculty members use the course management software. These tools typically include telephone contact with the helpdesk of the product provider and documentation, instruction, and/or listserves. Instructor Helpdesk tools may also enable faculty members to participate with other faculty in online discussion forums to share ideas or build knowledge.

Instructor Helpdesk tools often do not include assistance with content or instructional design.

Online Grading Tools

Online grading tools help instructors mark, provide feedback on student work, manage a gradebook.

Online Grading Tools enable instructors to mark assignments online, store grades, and delegate the marking process to teaching assistants. Some tools allow instructors to provide feedback to students, to export the gradebook to an external spreadsheet program, and to override the automatic scoring.

Student Tracking

Student Tracking is the ability to track the usage of course materials by students, and to perform additional analysis and reporting both of aggregate and individual usage.

Student Tracking tools include statistical analysis of student performance data and progress reports for individual students in the course. The progress reports generally consist of both activities and the time stamps of when the activity occurred.

3.2.2.3. Curriculum Design

Accessibility Compliance

Accessibility compliance means meeting the standards that allow people with disabilities to access information online. For example, the blind use a device called a screen reader to read the screen but Web pages need to be designed so that screen readers can read them.

Content Sharing/Reuse

Content sharing/reuse enables specific content created for one course to be conveniently shared with another instructor teaching a different course perhaps even at a different institution. Sometimes the content is in the form of learning objects. The system may enable sharing and reuse with a special file server or digital content repository that includes some form of digital rights management that spans campuses and even institutions.

Content sharing/reuse is a specialized form of digital publishing that is tailored to online learning situations. It is similar to the sharing and reuse of course templates that are stored centrally and used in more than one course, but different in that the content generally includes learning materials like lessons or learning objects and the access is managed centrally. There are several technically different variations including: content management systems, digital repositories, and content syndication systems. These systems are also similar to databases of content where the access to

specific content is managed with an authorization process that can protect the intellectual property.

Course Templates

Course templates are tools that help instructors create the initial structure for an online course.

Instructors use templates to go through a step-by-step process to set up the essential features of a course. Course Templates are artifacts of particular pedagogical approaches to instructional content and process. The local value of particular templates will depend in part on the match between the template designer's approach and the specific instructor's approach.

Curriculum Management

Curriculum management provides students with customized programs or activities based on prerequisites, prior work, or results of testing.

Curriculum Management includes tools to manage multiple programs, to do skills/competencies management, and to do certification management. These tools may be similar to the tools used in student services as part of providing academic advising to students.

Customized Look and Feel

Customized Look and Feel is the ability to change the graphics and how a course looks. This also includes the ability to institutionally brand courses.

Customized Look and Feel also includes the branding of content with institutional logos and navigation to provide a consistent look-and-feel across the entire institutional site and the integration of the system with additional institutional resources such as the library.

Instructional Design Tools

Instructional design tools help instructors creating learning sequences, for example, with lesson templates or wizards.

Instructional Standards Compliance

Instructional standards compliance concerns how well a product conforms to standards for sharing instructional materials with other online learning systems and other factors that may affect the decision whether to switch from this product to another.

Instructional Standards Compliance involves trying to make it possible for applications from different product producers to work well together. There are presently several proposed standards but the most prominent are the standards developed by the IMS Global Learning Consortium that define the technical specifications for interoperability of applications and services in distributed learning and support. The IMS standards can be found at www.imsproject org. The SCORM standards-in-progress integrate the industry specifications from IMS, AICC, IEEE, and ADRIANE and are operational standards with corresponding compliance test suites for learning objects (www.adlnet.org/main.html). In terms of compliance there appear to be three levels: awareness of the standards, claimed partial compliance, and self-tested compliance with the SCORM test suites. Other migration considerations are situations that would make switching to another application more complicated, such as proprietary data formats for content which make it difficult to import course content into another application. Also there are sometimes situations that complicate the upgrading from one version of the software to a later version. To the extent that student data is maintained in the system there can be separate complications in migrating non-course information to other versions or platforms.

3.3. Agents Usage in E-Learning and Course Management

There are many course management software tools ranging from home grown software environments to sophisticated commercial products. Some of these software use easy-to-use web authoring tools and some offer passive services. As a result instructors spend more time teaching a distance learning course than teaching the same course in class. This problem results mostly from the time consuming operational nature of the online courses. For example, instructor is expected to

regularly check the students' progress by visiting many web pages and using different tools within the course management system. This includes monitoring the message board activities log to verify student participation, consulting the drop box to see if students have submitted assignments and regularly visiting the course activity log to monitor the students' online activities. Performing these tasks in addition to handling hundreds of e-mail messages has become a major time consuming operation for most instructors. Intelligent agents functioning within the course management software system or a campus portal could perform some of these tasks, relieving the instructor from manual monitoring and management of course activities.

Until recently, a major requirement of course management software was ease of use. This no longer seems to be as important as before. We need smart learning environments that offer personal services with capabilities to learn, reason, have autonomy and be totally dynamic. Using intelligent agents in a course management environment can diminish some of the limitations of course management systems. For instance once a course instructor logs into the course environment, a teaching assistant agent could provide information such as the names of the students who have overdue assignments, have not collaborated in classroom message boards, have not taken an online quiz, or have not signed on for several days. Students' participation could even be ranked and categorized according to the instructor's preferences. The course instructor can configure an agent to give it autonomy to send personal e-mail to those who have done better than average or worse than expected.

To expand the capabilities of course management software, various kinds of intelligent agents that perform teaching and learning tasks in behalf of teachers and learners are being suggested by researchers[1] and are being developed in some research activities.[19] The proposed agents in the literature generally divides into three categories. Each group of agents is conceptualized to perform certain tasks normally carried out by instructors, students and administrators. Each group may consist of one or more intelligent agents focusing on certain tasks within a course site, a series of courses or the campus portal environment. These agents may communicate with their human clients using a combination of text, graphics, speech, facial expression, and voice recognition. Besides using the web browser on PC,

agents may use other types of communication environments including Personal Digital Assistants (PDAs), telephones, instant messenger systems etc. The following sections describe these three groups of agent usage in more detail.

3.3.1. Instructors' Agents

The instructors' intelligent agents assist the teacher in various teaching functions often performed by a human teaching assistant or a graduate student. It is a personal agent that may be configured by its owner, the human instructor. The concept is that the instructor will configure the digital TA at the beginning of the course. The configuration could include for instance, the agents' autonomy to send overdue notices to students on behalf of the teacher, and the language used in the body of the e-mail.

The digital TA is more useful in distance learning applications [1]. For instance, in a typical distance learning situation, the instructor is physically isolated form the students, not necessarily knowing if and when students worked on an assignment, for how long, or what types of collaboration they used. The teacher remains mostly unaware of the students' progress until an exam or until student submits and assignment or drops out of the course. In terms of student retention, the instructor ideally should be constantly and dynamically aware of a student's participation in a course and assist a discouraged student before he or she drops out. Additionally, a digital TA can assist a course instructor with course operation and maintenance similar to assistance of a human TA provides to an instructor. An example of instructor agent is the *inactivity agent* [1]. In this example, the agent is configured to send messages to the course instructor identifying students with more than one week of inactivity. The course instructor can further define the types and level of inactivity, such as lack of discussion on the class message board, failure to keep up with the reading assignments, or not taking quizzes. This is a very simple configuration of the agent.

In a more advanced configuration, the agent could continue monitoring student behavior after sending the initial notice to the student. An example of this might

include sending an additional notice with stronger language if the student continues to ignore the first or second messages. The agent may notify the course instructor about a potentially troubled student. With this notification to the course instructor, the agent could provide additional background information about each troubled student, including past submission record, grades, class ranking etc. This amount of information encourages the instructor to take quick and appropriate action for a troubled student.

As noted earlier, the Digital TA agents could include a series of agents, with only one being the inactivity agent illustrated in this example. The Cheat Buster intelligent agent described later is another useful example of an intelligent agent within the Digital TA group.

3.3.2. Tutor Agents

The intelligent agent acting as a Digital Tutor assists students with specific learning needs, just like a human tutor or a classmate. The Digital Tutor may act as a smart search engine, finding specific resources to solve learning needs for a student an intelligent agent that is expert both on content and on understanding a student's learning needs. Depending on the level of its sophistication, the Digital Tutor could "learn" and become more expert and useful as it provides more assistance to a student and receives more feedback. Consider an online distance-learning course where a student has difficulties understanding new learning objectives. The Digital Tutor has access to outside mobile agents who can help to identify appropriate resources. It is assumed that the Digital Tutor has access to students' learning profiles. Accessing student profiles and knowing students' strengths and weaknesses on a learning objective empowers the Digital Tutor to provide more useful resources. The student profile includes data dynamically collected from various databases, including campus information and registration databases (student information system, CMS databases and so on); personal preferences entered by an individual student; and usage data dynamically obtained by monitoring students' online activities. Examples of dynamic data obtained from various databases include the student's major and minor, previously taken courses, grades received for online

quizzes, and final transcript information. A smarter Digital Tutor may use assessment data from passed courses to make suggestions on new learning modules and information resources. An example of this scenario might be a student taking a second college English course who did very poorly in the grammar part of his first English course. Based on this data, the Digital Tutor might offer more learning exercises on grammar. A Digital Tutor may also act as a communication agent. Consider situations where students within a course are working on an online project. The communication agent can dynamically show the list of online students within the CMS environment who are working on the same project at the same time. Students can use this list to establish a virtual online communication and collaboration session with other online students in the classroom. The course chat room, instant messenger, or white-board can support this purpose. A student could further program the communication agent to inform him or her when another student in the same class working on the same assignment signs onto the CMS environment.

3.3.3. Digital Secretary

The intelligent agent acting as a Digital Secretary assists students and instructors in various logistical and administrative assistants needs. Like a human secretary, the Digital Secretary performs tasks as directed by its supervisor in this case, the human being at the keyboard. A simple example of the type of tasks that a Digital Secretary might perform is the "out of office" e-mail notification offered by Microsoft Outlook. The owner of a calendar can program Outlook to send an automatic e-mail notification to those who send e-mail messages during a specific time period. The Digital Secretary, however, should offer more intelligent and sophisticated services than the out-of-office agent. Consider a situation where an instructor would like to send a different auto-response e-mail to only those students taking a specific undergraduate course or those in the course that meets in the evenings. For instance, there might be only one group of Digital Secretary Agents within a student portal, while there might be a series of dedicated Digital TAs offered for each course. With this concept, the Digital Secretary can be accessed within the faculties' and students' portal environment, not within a course environment. An account owner of portal or course management software will configure the Digital

Secretary agent. Scheduling a meeting, finding a colleague with similar research interests, or finding the best math students who might serve as mentors are examples of tasks undertaken by a Digital Secretary in a teach-ing and learning environment. A Digital Secretary may also be used by other members of an educational institution who are not directly involved in teaching and learning, such as administrative staff, alumni, and parents.

3.3.4. Agents in Teaching and Learning Situations

Teaching and learning intelligent agents operate within CMS systems or campus portals. Each member of a CMS or campus portal (student, instructors, and others) has access to a series of personal intelligent agents after signing on. Users can configure their agents to perform specific tasks or services. The owner can program the agent to sequentially monitor certain incidences, compare them with preset thresholds, and perform certain tasks on the owner's behalf. For instance, a teacher could program his or her agent to send e-mail notification to students with a grade lower than C who additionally did not participate in the classroom message forum for the previous two weeks.

Depending on the type of agent, the access for configuring them could be located in the "*My Portal*" section of a campus portal or within a profile section of a CMS system. The agents could be multipurpose or course-specific (for example, an agent that monitors certain activities in Psychology 101). An agent may have access to a variety of dynamic and static data, including data obtained from the campus student information system, course management system, and student profile databases. Based on this information and configuration settings provided by the owner of an agent; the agent can think and perform intelligent actions. Given the massive amount of data processing involved, it might be necessary to run intelligent agent software on dedicated computer servers. Further-more, various tasks performed by an agent could be distributed among several computer servers.

3.3.5. Incorporating Agents in Learning Management Systems

Who should develop and build intelligent agents? How can the agents be integrated into the CMS and campus portals? How much will future intelligent learning environments cost, and what kind of resources and support will they require? These and other questions related to the design, development, integration, implementation, maintenance, and cost of intelligent learning environments will soon dominate the thinking of many technology administrators. As discussed earlier, intelligent agents can be integrated into existing teaching and learning environments as an add-on tool. Alternatively, CMS and portal vendors may improve the functionalities of various tools within their learning management systems to offer similar intelligent services. Consider the capabilities of the message board tool within the CMS system on your campus. The developer of the message board could release a newer version of its software that supports personalization and delivers user-defined functions, similar to the types of functions that external intelligent agents can perform. Campuses with self-built course management and portal software will have more flexibility in the design and integration of intelligent agents. Since they developed their own code and maintain ownership and control of their software, the in-house development of agents could be accomplished faster and easier. However, this is only feasible for larger institutions with greater programming and database expertise, substantial resources within IT support units, or more research groups within academic departments of the institution. Campuses that use off-the-self course management and portal software are at the mercy of their software providers to deliver intelligent learning tools. However, they might enjoy more cost-effective and plug-and-play situations. Additionally, integration and interoperability concerns are automatically resolved when integrating a vendor-designed agent into the learning management system developed by the same vendor Note that simple agents may not require a major development and implementation effort. For instance, the inactivity agent example given above can be developed easily using a few lines of code to access and analyze data already existing within a relatively few tables of course management or portal databases.

3.3.6. Hardware and Software Issues

Like CMS systems, agent designs rely heavily on the use of databases. Agents use external databases to obtain information about each user and local databases to store the query results and to build user profiles. Agents also use a substantial amount of computer resources on the database server side to run queries, stored procedures, triggers, and user-defined functions. Depending on the level of sophistication and intelligence, each agent may require its own server, operating system, and database software. This will certainly require budget provisions for purchasing new hardware and software, and for new maintenance and support services, especially in the area of database and data storage. Agent use makes it easy to forecast more applications of databases in our institutions once we begin delivering intelligent learning management systems.

CHAPTER 4

THE PROPOSED AGENT DEVELOPMENT PLATFORM SPECIFICATION

This section is concerned with building a software agent development and execution platform intended for integrated web based applications and gives detailed information about the technology and tools used in construction of the platform. The products and tools used in different layers and different platforms are explained separately. The proposed platform is intended to be used to develop agent applications running as *add-on* utilities for existing integrated systems sharing common data resources and running on different platforms. This chapter gives brief information about the infrastructure for those systems as well as the agent running environment.

4.1. Software Agent Environment

The software agent add-on utilities are developed with Java in JADE; Java Agent Development Environment. The agent system uses Java Runtime Environment version 1.4 and uses the common databases with its master software which is operating on a Microsoft SQL Server 2000. The JADE environment is explained in detail in the following section. The agent system is developed in Sun ONE Studio 5.0 SE.

4.1.1. Java Agent Development Environment (JADE)

JADE is the middleware developed by TILAB for the development of distributed multi-agent applications based on the peer-to-peer communication architecture [20]. The intelligence, the initiative, the information, the resources and

the control can be fully distributed on mobile terminals as well as on computers in the fixed network. The environment can evolve dynamically with peers, which in JADE are called agents, which appear and disappear in the system according to the needs and the requirements of the application environment. Communication between the peers, regardless of whether they are running in the wireless or wired network, is completely symmetric with each peer being able to play both the initiator and the responder role. JADE is fully developed in Java and is based of the following driving principles:

- Interoperability – JADE is compliant with the FIPA specifications [7]. As a consequence, JADE agents can interoperate with other agents, provided that they comply with the same standard.
- Uniformity and portability – JADE provides a homogeneous set of APIs that are independent from the underlying network and Java version. More in details, the JADE run-time provides the same APIs both for the J2EE, J2SE and J2ME environment. In theory, application developers could decide the Java run-time environment at deploy-time.
- Easy to use – The complexity of the middleware is hidden behind a simple and intuitive set of APIs.
- Pay-as-you-go philosophy – Programmers do not need to use all the features provided by the middleware. Features that are not used do not require programmers to know anything about them, neither adds any computational overhead.

4.1.1.1. The Architectural model

JADE includes both the libraries (i.e. the Java classes) required to develop application agents and the run-time environment that provides the basic services and that must be active on the device before agents can be executed. Each instance of the JADE run-time is called container (since it "contains" agents). The set of all containers is called platform and provides a homogeneous layer that hides to agents (and to application developers also) the complexity and the diversity of the underlying tires (hardware, operating systems, types of network, JVM).

As depicted in Figure 4.1., JADE is compatible with the J2ME CLDC/MIDP1.0 environment. It has already been tested on the fields over the GPRS network with different mobile terminals among which: Nokia 3650, Motorola Accompli008, Siemens SX45, PalmVx, Compaq iPaq, Psion5MX, HP Jornada 560. The JADE run-time memory footprint, in a MIDP1.0 environment, is around 100 KB, but can be further reduced until 50 KB using the ROMizing technique [21], i.e. compiling JADE together with the JVM. JADE is extremely versatile and therefore, not only it fits the constraints of environments with limited resources, but it has already been integrated into complex architectures such as.NET or J2EE [22] where JADE becomes a service to execute multi-party proactive applications. The limited memory footprint allows installing JADE on all mobile phones provided that they are Java-enabled.

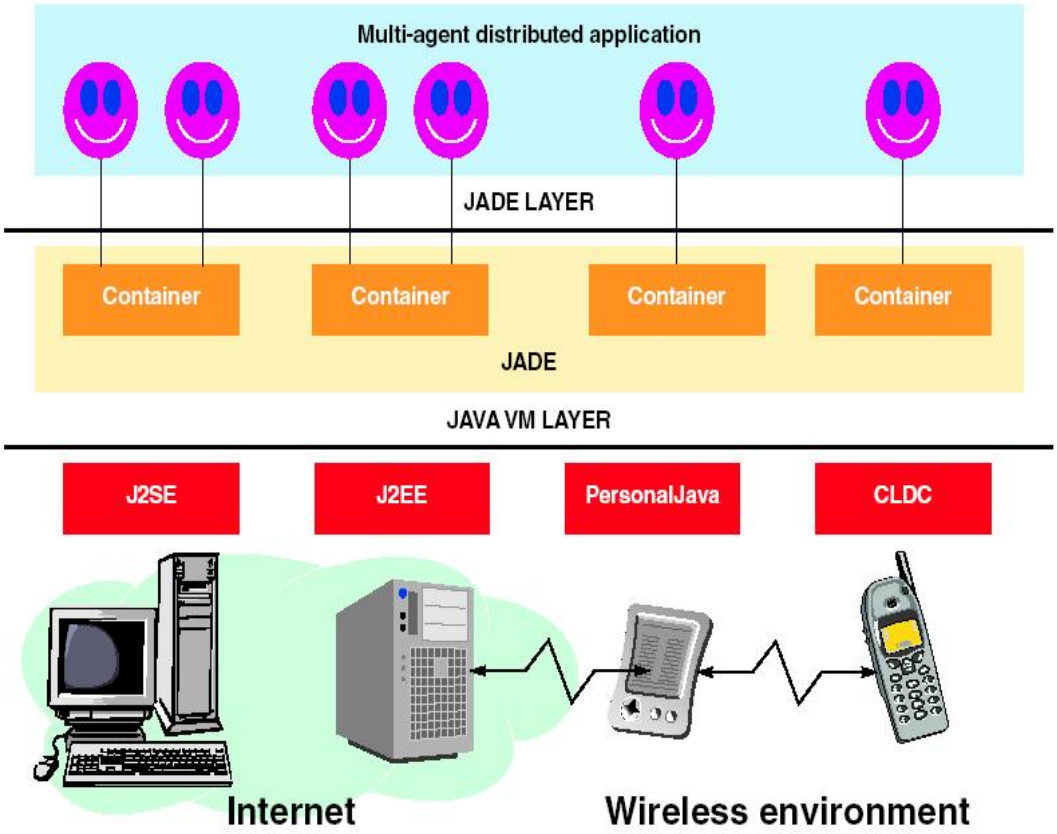

Figure 4.1 The Jade architecture

4.1.1.2. The Functional Model

From the functional point of view, JADE provides the basic services necessary to distributed peer-to peer applications in the fixed and mobile environment. JADE allows each agent to dynamically discover other agents and to communicate with them according to the peer-to-peer paradigm. From the application point of view, each agent is identified by a unique name and provides a set of services. It can register and modify its services and/or search for agents providing given services, it can control its life cycle and, in particular, communicate with all other peers. Agents communicate by exchanging asynchronous messages, a communication model almost universally accepted for distributed and loosely-coupled communications1, i.e. between heterogeneous entities that do not know anything about each other. In order to communicate, an agent just sends a message to a destination. Agents are identified by a name (no need for the destination object reference to send a message) and, as a consequence, there is no temporal dependency between communicating agents. The sender and the receiver could not be available at the same time. The receiver may not even exist (or not yet exist) or could not be directly known by the sender that can specify a property (e.g. “all agents interested in football”) as a destination. Because agents identifies each other by their name, hot change of their object reference are transparent to applications. Despite this type of communication, security is preserved, since, for applications that require it, JADE provides proper mechanisms to authenticate and verify “rights” assigned to agents. When needed, therefore, an application can verify the identity of the sender of a message and prevent actions not allowed to perform (for instance an agent may be allowed to receive messages from the agent representing the boss, but not to send messages to it). All messages exchanged between agents are carried out within an envelope including only the information required by the transport layer. That allows, among others, to encrypt the content of a message separately from the envelope. The structure of a message complies with the ACL language defined by FIPA [7] and includes fields, such as variables indicating the context a message refers-to and timeout that can be waited before an answer is received, aimed at supporting complex interactions and multiple parallel conversations. To further support the implementation of complex conversations, JADE provides a set of skeletons of typical interaction patterns to perform specific tasks, such as negotiations, auctions

and task delegation. By using these skeletons (implemented as Java abstract classes), programmers can get rid of the burden of dealing with synchronization issues, timeouts, error conditions and, in general, all those aspects that are not strictly related to the application logic. To facilitate the creation and handling of messages content, JADE provides support for automatically converting back and forth between the format suitable for content exchange, including XML and RDF, and the format suitable for content manipulation (i.e. Java objects). This support is integrated with some ontology creation tools, e.g. Protégé, allowing programmers to graphically create their ontology. JADE is opaque to the underlying inference engine system, if inferences are needed for a specific application, and it allows programmers to reuse their preferred system. It has been already integrated and tested with JESS and Prolog. To increase scalability or also to meet the constraints of environments with limited resources, JADE provides the opportunity of executing multiple parallel tasks within the same Java thread. Several elementary tasks, such as communication, may then be combined to form more complex tasks structured as concurrent Finite States Machines. In the J2SE and Personal Java environments, JADE supports mobility of code and of execution state. That is, an agent can stop running on a host, migrate on a different remote host (without the need to have the agent code already installed on that host), and restart its execution from the point it was interrupted (actually, JADE implements a form of not-so-weak mobility because the stack and the program counter cannot be saved in Java). This functionality allows, for example, distributing computational load at runtime by moving agents to less loaded machines without any impact on the application. The platform also includes a naming service (ensuring each agent has a unique name) and a yellow pages service that can be distributed across multiple hosts. Federation graphs can be created in order to define structured domains of agent services. Another very important feature consists in the availability of a rich suite of graphical tools supporting both the debugging and management/monitoring phases of application life cycle. By means of these tools, it is possible to remotely control agents, even if already deployed and running: agent conversations can be emulated, exchanged messages can be sniffed, tasks can be monitored, and agent life-cycle can be controlled. The described pieces of functionality, and particularly the possibility of remotely activating (both from code and from console), even on mobile terminals, tasks, conversations and new peers,

makes JADE very well suited to support the development and execution of distributed, machine-to-machine, multi-party, intelligent and proactive applications.

4.1.1.3. JADE in the Mobile Environment

As already mentioned, the JADE run-time can be executed on a wide class of devices ranging from servers to cell phones, for the latter the only requirement being Java MIDP1.0 (or higher versions). In order to properly address the memory and processing power limitations of mobile devices and the characteristics of wireless networks (GPRS in particular) in terms of bandwidth, latency, intermittent connectivity and IP addresses variability, and at the same time in order to be efficient when executed on fixed network hosts, JADE can be configured to adapt to the characteristics of the deployment environment. JADE architecture, in facts, is completely modular and, by activating certain modules instead of others, it is possible to meet different requirements in terms of connectivity, memory and processing power. More in details, a module called LEAP allows optimizing all communication mechanisms when dealing with devices with limited resources and connected through wireless networks. By activating this module, a JADE container is "split", into a front-end, actually running on the mobile terminal, and a back-end running in the fixed network. A proper architectural element, called mediator, must be already active and is in charge of instantiating and holding the back-ends (that basically are entries in the mediator itself). To face work-load problems it is possible to deploy several mediators each one holding several back ends. Each front-end is linked to its corresponding back-end by means of a permanent bi-directional connection. It is important to note that there is no difference at all for application developers depending on whether an agent is deployed on a normal container or on the front-end of a split container, since both the available functionality and the APIs to access them are exactly the same. The approach has a number of advantages:

- Part of the functionality of a container is delegated to the back-end, thus making the front-end extremely lightweight in terms of required memory and processing power.
- The back-end masks, to other containers, the actual IP address assigned to the wireless device and, among the others, allows hiding to the rest of the multi-agent

system a possible change of IP address. • The front-end is able to detect a loss of connection with the back-end (for instance due to an out of coverage condition) and re-establish it as soon as possible.

- Both the front-end and the back-end implement a store-and-forward mechanism: messages that cannot be transmitted due to a temporary disconnection are buffered and delivered as soon as the connection is re-established.
- Several information that containers exchange (for instance to retrieve the container where an agent is currently running) are handled only by the back-end. This approach, together with a bit-efficient encoding of communications between the front-end and the backend, allows optimizing the usage of the wireless link.

4.1.1.4. Technical/functional Characteristics

Following is a list of technical and functional characteristics of JADE:

- Distributed, multi-party application with peer-to-peer communication.
- Compliance with the FIPA standard.
- Agent life cycle management.
- White pages and yellow pages services with the opportunity of creating federation graphs at run-time.
- Graphical tools supporting the debugging, management and monitoring phases.
- Support for agent code and execution state migration.
- Support for complex interaction protocols (e.g. contract-net).
- Support for messages content creation and management including XML and RDF.
- Support for integration in JSP pages by means of a tag library.
- Support for application level security (currently only in J2SE).
- Transport protocols selectable at run-time. Currently available: JAVA-RMI, JICP (JADE proprietary protocol), HTTP and IIOP.

4.1.1.5. The open source project

The whole JADE source code is distributed under an open source policy, the Lesser GNU Public License (LGPL for short) [23]. LGPL enables full exploitation of JADE, even in a business environment, while enforcing the constraint that any modification of JADE source code and any derivative work be returned to the community under the LGPL license itself. No restrictions, instead, are put on applications and other categories of software that uses JADE. A large user base, counting more than a thousand members, gathered around this project; many among them are contact points within their company or university, bridging internal JADE users with the worldwide community. Community subscribers come partly from academic environments (JADE is very popular as a teaching support environment in distributed AI courses), partly from R&D centers of world leading companies such as Motorola, HP, Siemens and Rockwell Automation, and partly from small start-ups such as Mobile Tribe and Acklin, looking at JADE as an enabling technology for their businesses. Outstanding contributions of Motorola, Siemens, and Broadcom have to be acknowledged because, within the framework of the LEAP IST project [7], they strongly contributed to port the JADE platform to the J2ME/MIDP environment. The JADE project is supported by a web site [24] where users can download code and documentation, report possible bugs and browse a collection of useful links maintained by the JADE team. Moreover, two mailing lists are available to developers for discussing technical issues or just for staying tuned about the project, e.g. to be informed about new releases. Due to such an active user base, hundreds of JADE downloads were registered in peak days and the project counts now more than 40,000 downloads in total. JADE welcomes contributions of the Open Source Community that can be given under different forms: simply making publicly known the usage of JADE, reporting and, better, fixing bugs and documentation, replying and giving support to less-experienced users on the mailing list, contributing with new add-ons and software modules. The JADE Governing Board In May 2003 TILAB and Motorola launched a new initiative, the JADE Governing Board, a not-for-profit organization, with the intent of promoting the evolution and the adoption of JADE by the mobile telecommunications industries as

a java-based de-facto standard middleware for agent-based applications in the mobile personal communication sector. The mission of the JADE Governing Board is the industrial affirmation of JADE through the establishment of consensus and the contribution of key players in the mobile sector in order to expand consumer options and interest through new wireless agent applications. The JADE-board paves the way for mobile VAS services, where peer-to-peer communication and services on PCs, PDA's and phones will enable tailored solutions for the mobile users and mobile teams to meet the increasing demand for intelligent mobile lifestyles. The Board intends to leverage, continue and consolidate the Open-source tradition through the continuous support and involvement of the JADE Open-source Community. The Board has been formed as a contractual consortium among the Members, it is open and it welcomes all those companies and organizations that have a concrete business interest in the extension of JADE and that commit to contribute to its development and promotion. The JADE Web site provides information on how to join the Board.

4.1.2. Application Features of JADE

JADE is a middleware that simplifies the development of agent applications. Several companies are already using it for very different application sectors including supply chain management, manufacturing, rescue management, fleet management, auctions, tourism, etc. The following sections try to describe which application features best benefit from JADE.

4.1.2.1. Distributed Applications Composed of Autonomous Entities

First of all, JADE simplifies the development of distributed applications composed of autonomous entities that need to communicate and collaborate in order to achieve the working of the entire system. A software framework that hides all complexity of the distributed architecture is made available to application developers, who can focus their software development just on the logic of the application rather than on middleware issues, such as discovering and contacting the entities of the system. This type of distributed applications enabled by JADE, in

particular when applied to the mobile environment, ignite a new trend of evolution that we like to name smart-device smart-interconnection: the software on each device is equipped with autonomy, intelligence, and capability of collaboration and the value of the system is given by the capabilities of the devices and by their interaction and collaboration capabilities. This is quite different from the ubiquitous access trend where the value of the system is given by the content and the capability of accessing the content from anywhere.

4.1.2.2. Negotiation and Coordination

JADE simplifies the development of applications that require negotiation and coordination among a set of agents, where the resources and the control logics are distributed in the environment. In fact, easy-to-use software libraries to implement peer-to-peer communication and interaction protocols (i.e. patterns of interaction between autonomous entities) are provided by JADE to developers.

4.1.2.3. Proactivity

JADE agents control their own thread of execution and, therefore, they can be easily programmed to initiate the execution of actions without human intervention just on the basis of a goal and state changes. This feature, that is usually called proactivity, makes JADE a suitable environment for the realization of machine-to-machine (m2m) applications, for example, for industrial plant automation, traffic control and communication network management.

4.1.2.4. Multi-party applications

Peer-to-peer architectures are more efficient than client-server architectures for developing multi-party applications, as the server might become the bottleneck and the point of failure of the entire system. Because JADE agents can both provide and consume services, they remove any need to distinguish between clients and servers.

JADE agents allow clients to communicate each-other without the intervention of a central server. Moreover, the fact that intelligence, information and control are distributed, allows the realization of applications where the ownership is distributed among the peers (agents) given that each peer may be able, and authorized to perform, just a subset of the actions of the application.

4.1.2.5. Interoperability

JADE complies with the FIPA specifications that enable end-to-end interoperability between agents of different agent platforms. All applications where inter-organization communication is needed can benefit from interoperability, including machine-to-machine and manufacturing.

4.1.2.6. Openness

JADE is an open-source project that involves the contributions and collaborations of the user community. This user-driven approach allows both users and developers to contribute with suggestions and new code, which guarantees openness and usefulness of the APIs. Of course, anarchy must be avoided and the JADE Governing Board is the actor that formally controls the evolution of JADE in terms of new APIs and functionalities.

4.1.2.7. Versatility

JADE provides a homogeneous set of APIs that are independent from the underlying network and Java version. It in fact provides the same APIs both for the J2EE, J2SE and J2ME environment. This feature allows application developers to reuse the same application code both for a PC, a PDA or a Javaphone; it allows postponing this choice as late as possible, in theory, until the deploy-time.

4.1.2.8. Ease of Use and Mobile Applications

JADE API's are easy to learn and use. JADE has been designed to simplify the management of communication and message transport by making transparent to the developer the management of the different communication layers used to send a message from an agent to another agent, and so allowing her/him to concentrate on the logic of the application. Of course, the effect of this feature is to make faster the development of applications. JADE reduces the application development time in respect to the time necessary to develop the same application by using only Java standard packages. In particular when developing distributed applications for mobile terminals, JADE APIs and ready-to-use functionalities allow to strongly reduce the application development time and costs (some estimations have been given that indicates reduction of development time up to 30%).

4.2. Infrastructure of Integrated Software

The applications that can benefit from our software agent system may be composed of different technologies and different types of tools. The example agent application currently running on our platform is sharing resources with a variety of different software products including Java applications, Microsoft .NET applications and also old versions of ASP technologies together. Their common point is; all of them use the same data resources. Following sections deeply concern with the different technologies that can be involved within the proposed platform.

4.2.1. Microsoft .NET Framework

The .NET Framework is a new way to expose operating system and other APIs. For years, the set of Windows functionality that was available to developers and the way that functionality was invoked were dependent on the language environment being used. For example, the Windows operating system provides the ability to create windows (obviously). Yet, the way this feature was invoked from a C++ program was dramatically different from the way it was invoked from a Visual

Basic program. With .NET, the way that operating system services are invoked is uniform across all languages (including code embedded in ASP.NET pages). This portion of .NET is commonly referred to as the .NET Framework class library. [25]

The .NET Framework is a new infrastructure for managing application execution. To provide a number of sophisticated new operating-system services—including code-level security, cross-language class inheritance, cross-language type compatibility, and hardware and operating-system independence, among others—Microsoft developed a new runtime environment known as the Common Language Runtime (CLR). The CLR includes the Common Type System (CTS) for cross-language type compatibility and the Common Language Specification (CLS) for ensuring that third-party libraries can be used from all .NET-enabled languages. To support hardware and operating-system independence, Microsoft developed the Microsoft Intermediate Language (MSIL, or just IL). IL is a CPU-independent machine language-style instruction set into which .NET Framework programs are compiled. IL programs are compiled to the actual machine language on the target platform prior to execution (known as just-in-time, or JIT, compiling). IL is never interpreted.

The .NET Framework is a new web server paradigm. To support high-capacity web sites, Microsoft has replaced its Active Server Pages (ASP) technology with ASP.NET. While developers who are used to classic ASP will find ASP.NET familiar on the surface, the underlying engine is different, and far more features are supported. One difference is that ASP.NET web page code is now compiled rather than interpreted, greatly increasing execution speed.

The .NET Framework is a new focus on distributed-application architecture. Visual Studio .NET provides top-notch tools for creating and consuming web services vendor-independent software services that can be invoked over the Internet. The .NET Framework is designed top to bottom with the Internet in mind. For example, ADO.NET, the next step in the evolution of Microsoft's vision of "universal data access," assumes that applications will work with disconnected data by default. In addition, the ADO.NET classes provide sophisticated XML capabilities, further increasing their usefulness in a distributed environment.

4.2.2. Microsoft .NET Development Platform

Visual Studio .NET is an environment for developing Windows and Web applications. Visual Basic .NET is just one of the languages we can use to program our applications. Actually, Visual Studio .NET was designed to host any language, and many companies are working on languages that will be integrated in Visual Studio .NET. Some people will develop Windows applications in Visual Studio .NET with COBOL, or FORTRAN. So, what's the distinction between Visual Studio .NET and the language? Visual Studio .NET is the environment that provides all the necessary tools for developing applications. The language is only one aspect of a Windows application. The visual interface of the application isn't tied to a specific language, and the same tools we will use to develop our application's interface will also be used by all programmers, regardless of the language they'll use to code the application. The tools we'll use to access databases are also independent of the language. Visual Studio provides tools that allow us to connect to a database; inspect its objects, retrieve the information we're interested in, and even store it in objects that can be accessed from within any language. There are many visual tools in the IDE, like the Menu Designer. This tool allows us to visually design menus and to set their names and basic properties (such as checking, enabling, or disabling certain options). Designing a menu doesn't involve any code, and it's carried out with point-and click operations. Of course, we will have to insert some code behind the commands of our menus, and (again) we can use any language to program them. To simplify the process of application development, Visual Studio .NET provides an environment that's common to all languages, which is known as integrated development environment (IDE). The purpose of the IDE is to enable the developer to do as much as possible with visual tools, before writing code. The IDE provides tools for designing, executing, and debugging our applications. [25]

4.2.3. Visual Basic .NET

Visual Basic .NET is the next generation of Visual Basic, but it is also a significant departure from previous generations. However, Microsoft has made some changes to make Visual Basic .NET a better language and an equal player in the

.NET world. These include such additions as a Class keyword for defining classes and an Inherits keyword for object inheritance, among others. Visual Basic 6 code can't be compiled by the Visual Basic .NET compiler without significant modification. [26]

Visual Basic .NET is released shortly after the tenth anniversary of the first version of VB. The original language that changed the landscape of computing has lasted for 10 years and has enabled more programmers to write Windows application than any other language. In the world of computing, however, things change very fast, including languages. At some point, they either die, or they evolve into something new. Visual Basic was a language designed primarily for developing Windows applications. It was a simple language, because it managed to hide many of the low-level details of the operating system. Those who wanted to do more with Visual Basic had to resort to Windows API. In a way, earlier versions of Visual Basic were ‘sandboxed’ to protect developers from scary details.

Microsoft had to redesign Visual Basic. The old language just didn’t belong in the .NET picture (at least, it wouldn’t integrate very well into the picture). Visual Basic .NET is not VB7; it’s a drastic departure from VB6, but a necessary departure.

4.2.4. ASP.NET

ASP.NET is a key part of the wider Microsoft .NET initiative, Microsoft’s new application development platform. .NET is both application architecture to replace the Windows DNA model and a set of tools, services, applications and servers based around the .NET Framework and common language runtime (CLR). Rather than just being ASP 4 or an incremental upgrade, ASP.NET is a complete rewrite from the ground up, using all the advanced features .NET makes available. ASP.NET can take advantage of all that .NET has to offer, including support for around 20 or more .NET languages from C# to Perl.NET, and the full set of .NET Framework software libraries. Web applications written in ASP.NET are fast, efficient, manageable, scalable, and flexible, but, above all, easy to understand and to code! Components and Web applications are all compiled .NET objects written in the same languages,

and they offer the same functionality, so no need to leave the ASP environment for purely functional reasons. With a few lines of code, ASP.NET can talk to XML, serve as or consume a Web service, upload files, “screen scrape” a remote site, or generate an image.

With the .NET Framework and ASP.NET, Microsoft has not just shown itself to be a contender in Web development technologies, but many commentators also believe Microsoft has taken the lead.ASP.NET is well equipped for any task you want to put to it, from building intranets to e-business or e-commerce sites. Microsoft has been very careful to include the functionality and flexibility developers will require, while maintaining the easy-to-use nature of ASP.

- With ASP.NET you now have a true choice of languages. All the .NET languages have access to the same foundation class libraries, the same type of systems, equal object orientation and inheritance abilities, and full interoperability with existing COM components.
- You can use the same knowledge and code investment for everything from Web development to component development or enterprise systems, and developers do not have to be concerned about differences in APIs or variable type conversions, or even deployment.
- ASP.NET incorporates all the important standards of our time, such as XML and SOAP, plus with ADO.NET and the foundation class libraries, they are arguably easier to implement than in any other technology, including Java.
- An ASP.NET programmer still only needs a computer with Notepad and the ability to FTP to write ASP code, but now with the .NET Framework command-line tools and the platform’s XML-based configuration, this is truer than before!
- Microsoft has included in the .NET Framework an incredibly rich feature set of library classes, from network-handling functions for dealing with Transmission Control Protocol/Internet Protocol (TCP/IP) and Domain Name System (DNS), through to XML data and Web Services, to graphic drawing.
- In the past, the limitations of ASP scripting meant components were required for functionality reasons, not just for architectural reasons. ASP.NET has access to

the same functionality and uses the same languages in which you would create components, so now components are an architectural choice only.

- A .NET developer is shielded from changes in the underlying operating system and API, as the .NET technologies deal with how your code is implemented; and with the Common Type System, you don't have to worry whether the component you are building uses a different implementation of a string or integer to the language it will be used in.

4.3. Patterns and Methodologies Used in Development Process

In developing Course ON-LINE, we followed the Extreme Programming software development model [32,33]. The design and development of the software has lasted about 4 man/months and developed with 2 programmers building a pair. The concept of pair programming is explained later in this chapter. The rapidness in development is achieved thanks to the method XP answers the changing requirements. The used development processes and methods are the key to this short project size. In the following section, the extreme programming and used practices are explained in more detail.

4.3.1. Extreme Programming

Extreme Programming has some very obvious advantages when compared to the traditional approaches.

We discussed the definition and characteristics of a process above. When the process involves building a software product, it is called a software lifecycle, because it describes the life of a software product from its conception to its implementation, delivery, use and maintenance. OD-STD-2167A/498, the current prevailing standard guiding software development, has been interpreted as mandating as specific process for use on all military acquisitions. This process is represented by the "Waterfall Model", which serves as the conceptual guideline for almost all Air Force and NASA software development. The waterfall model was first describer by

Royse [34]. The waterfall model is an activity-centered life cycle model that prescribes a sequential execution of a subset of the development processes and management processes. The requirement activities are all completed before the systems design activity starts. The goal is to never turn back once an activity is completed. The key feature of his model is the constant verification activity that ensures that each development activity does not introduce unwanted or deletes mandatory requirements. Many of the phases require successful completion of a government review process. Critics of the "Waterfall" Model, in fact, find that the model is geared to recognize documents as a measure of progress rather than actual results.

The nine major activities described in 2167A/498 are as follows:

1. Systems Concept/System Requirements Analysis
2. Software Requirements Analysis
3. Software Parametric Cost Estimating
4. Preliminary Design
5. Detailed Design
6. Coding and Computer Software unit (CSU) Testing
7. Computer Software Component (CSC) Integration and Testing
8. Computer Software Configuration Item (CSCI) Testing
9. System Integration and Operational Testing

As a response to rapid-change in requirements and the increase of failures in software development projects due to the incapability of traditional software engineering approaches like the waterfall model above to this change, new agile development methodologies are introduced. Extreme Programming (XP) is one of those methodologies. It is described as “a lightweight discipline of software development, which is designed for use with small teams who need to develop software quickly in an environment of rapidly-changing requirements, based on principles of simplicity, communication, feedback, and courage.” Most significant and major difference of XP from traditional software engineering methodologies is; XP focuses on the product whereas process is focused by many of the other software development methodologies. XP is based on rapid application development and it is informal and verbal. Change in requirements is not considered as a problem. XP is

based on some practices that are easy to follow but not strict and formal rules. XP Practices are grouped in four; Planning Practices, Designing Practices, Coding Practices and Testing Practices [33].

XP improves a software project in four essential ways; communication, simplicity, feedback, and courage. XP programmers communicate with their customers and fellow programmers. They keep their design simple and clean. They get feedback by testing their software starting on day one. They deliver the system to the customers as early as possible and implement changes as suggested. With this foundation XP programmers are able to courageously respond to changing requirements and technology.

The major distinction between XP and other approaches is that; XP focuses on the product whereas traditional SE approaches focus on the process. The idea is to develop hundred percent working software products no matter how you develop it. The process is defined with practices, which are fruitful and easy to follow. Other major distinction is; changes in the requirements are welcomed at any stage in an XP project but change in the requirements is controlled and avoided in other SE methodologies.

McCormick compares the two approaches as given in Table 4.1. [35]

As seen from the table, XP is informal and verbal, is based on rapid application development, and change is not considered a problem, whereas software engineering is formal and written, and change is controlled and avoided. Software engineering attempts to complete design before starting coding whereas XP does not devote much effort to design, and code itself is considered to be the design. In short, XP focuses on programming and on directly producing the code.

XP proves to be very useful and productive especially in cases where requirements change frequently and where architectural issues are already solved either because of the nature of the application or because of the tools and the platforms chosen.

Table 4.1 Traditional Software Engineering Approaches vs. XP

Software Engineering	Extreme Programming
Avoid change in coding phase	Code change is no problem
Specifications must be formal and written	Specifications may be informal and verbal
Design must be completed before coding	Code is the design
Communication is a problem for programmers	Programmers can communicate well
Change control should be enforced for requirements	Informal requirements suffice
Rapid Application Development is avoided	Rapid Application Development is favoured

Apart from the core XP practices we used some practices introduced in [36]. These are:

- Issue-Based Programming is a designing practice where issues determine priorities. At any time instance during the software development project there are many issues to be resolved and many issues already resolved. At the early stages of development, most of the issues identified are related to requirements rather than design or implementation, whereas at later stages most issues are related to implementation. Still there are implementation-related issues at early stages and requirements-related issues at later stages. An issue table is maintained throughout the software development project and issues are identified as closed when they are resolved. A closed issue may become open later. The prioritization of issues is done regarding the requests of the customer and by negotiation between the customer and the development team.

- Comment-first coding is a coding practice. It breaks the coding of a single module into two phases: coding the semantics, and coding the syntax. When coding a module, first the algorithm of that code is written to the editor as

comment lines in natural language. The level of detail should be such that the architecture of the algorithm should be understood at a glance and that converting each comment line to the valid syntax of the language used is easy for all the programmers in the team. After the first phase is completed, the output is a formatted text, which is easy to read and easy to convert to programming language's syntax. In the second phase, the programmer starts to code each construct following an outside-in approach. The reason why we call outside-in but not top-down is that the level of granularity is more or less the same at the end of the two phases.

- JIT collective code ownership is an extension of collective code ownership, which is an existing coding practice. On an XP project, any pair of programmers can improve any code at any time. This is called collective code ownership. This means each programmer is aware of everything in the project and each programmer is responsible of all parts of the code. This kind of a practice leads to efficiency in some cases such as when a programmer is waiting for a piece of code to be written by another programmer that is responsible of that code. On the other hand, the overhead of being aware of everything for each single programmer may be very high in many cases. To overcome this problem, XP teams follow a common coding standard, so that all the code looks as if it is written by a single programmer.

4.3.2. Object – Oriented Programming

While developing Course ON-LINE and GAIA add-on, the development team used an object – oriented programming language and had considerable benefits.

Object – oriented method is a software design method that models the characteristics of abstract or real objects using classes and objects. In procedural programming languages, programming tends to be action – oriented, and the unit of programming is the function. In object – oriented programming languages the unit of programming is the class from which objects are eventually instantiated. There are many definitions of an object, such as found in [37]: "An object has state, behavior,

and identity; the structure and behavior of similar objects are defined in their common class; the terms instance and object are interchangeable". This is a "classical languages" definition, as defined in [38] where "classes play a central role in the object model", since they do not in prototyping/delegation languages. "The term object was first formally applied in the Simula language, and objects typically existed in Simula programs to simulate some aspect of reality" [39]. Other definitions referenced by Booch include Smith and Tockey: "an object represents an individual, identifiable item, unit, or entity, either real or abstract, with a well-defined role in the problem domain." and "anything with a crisply defined boundary" (in context, this is "outside the computer domain". A more conventional definition appears on pg 44). Booch goes on to describe these definitions in depth. [40] defines: "An "object" is anything to which a concept applies", and "A concept is an idea or notion we share that applies to certain objects in our awareness". [41] defines: "We define an object as a concept, abstraction or thing with crisp boundaries and meaning for the problem at hand." [42] defines: "An object is an abstraction of a set of real-world things such that:

• All of the real-world things in the set - the instances - have the same characteristics

• All instances are subject to and conform to the same rules"

And on identifying objects: "What are the *things* in this problem? Most of the things are likely to fall into the following five categories: Tangible things, Roles, Incidents, Interactions, and Specifications." [41] covers "Identifying Key Abstractions" for objects and classes based on an understanding of the problem domain and [42] provides a novel approach to identifying objects through use-cases (scenarios), leading to a use-case driven design. Use cases have become very important and popular today, providing an easy way to identify objects and methods by their use in satisfying system requirements and uses. Uses cases occur throughout the development lifecycle.

The basic idea behind an object is that of simulation. Most programs are written with very little reference to the real world objects the program is designed to work with; in object oriented methodology, a program should be written to simulate the states and activities of real world objects. This means that apart from looking at data structures when modeling an object, we must also look at methods associated

with that object, in other words, functions that modify the objects attributes. A method is an operation which can modify an objects behavior. In other words, it is something that will change an object by manipulating its variables.

A class is a blueprint for an object. What this basically means is that we provide a blueprint, or an outline of an object. This blueprint is valid whether we have one or one thousand such objects. A class does not represent an object; it represents all the information a typical object should have as well as all the methods it should have. A class can be considered to be an extremely extended TYPE declaration in the C programming language, since not only are variables held but methods too.

Object Oriented Programming method has a very distinguishing feature: Inheritance and sub-classes. A subclass is a class definition which derives functionality from another class definition. Inheritance provides a natural classification for kinds of objects and allows for the commonality of objects to be explicitly taken advantage of in modeling and constructing object systems. Natural means we use concepts, classification, and generalization to understand and deal with the complexities of the real world. See the example below using computers.

Inheritance is a relationship between classes where one class is the parent (base/superclass/ancestor/etc.) class of another. Inheritance provides programming by extension (as opposed to programming by reinvention and can be used as an is-a-kind-of (or is-a) relationship or for differential programming. Multiple Inheritance occurs when a class inherits from more than one parent/superclass. For example, a person is a mammal and an intellectual_entity, and a document may be an editable_item and a kind of literature.

So why user inheritance? Inheritance is a natural way to model the world or a domain of discourse, and so provides a natural model for OOA and OOD (and even OOP). This is common in the AI domain, where semantic nets use inheritance to understand the world by using classes and concepts for generalization and categorization, by reducing the real-world's inherent complexity.

Inheritance also provides for code and structural reuse. In the above Computer class diagram, all routines and structure available in class Computer are available to all subclasses throughout the diagram. All attributes available in Personal computers are also available to all of its subclasses. This kind of reuse takes advantage of the is-a-kind-of relationship. Class libraries also allow reuse between applications, potentially allowing order-of-magnitude increases in productivity and reductions in defect rates (program errors), as library classes have already been tested and further use provides further testing providing even greater reliability.

With differential programming, a class does not have to be modified if it is close to what's required; a derived class can be created to specialize it. This avoids code redundancy, since code would have to be copied and modified otherwise.

Polymorphism is often explicitly available in many Object Oriented languages (such as C++, CLOS, Eiffel, etc.) based on inheritance when type and class are bound together (typing based on subclassing, or subclass polymorphism), since only an object which is a member of (inherits from) a class is polymorphically assignment compatible with (can be used in place of) instances or references of that class. Such assignment can result in the loss of an object's dynamic type in favor of a static type (or even loss of an object's representation to that of the static class, as in C++ slicing). Maintaining the dynamic type of objects can be provided (and preferred); however, C++ provides both sliced and non-sliced replacement in a statically typed environment.

Object Oriented Programming with all of the above features increases code reusability. Course ON-LINE received great benefit from this feature. Most of the business logic was encapsulated and reused throughout the code. The same business modules were use in the web tier and in the desktop tier. Using the advantages of inheritance, the data model was mapped to an object model, forming a data access layer and an abstraction model. All of the data access was done through this layer, giving a DBMS vendor independent code.

4.3.3. Software Design Pattern Used

With usage of software design patterns, a framework was constructed during the development of the course management system Course ON-LINE. In this section we take a closer look on these patterns and how they affected the development of the Course ON-LINE. These patterns can be found in [43].

4.3.3.1 Model-View-Controller

The purpose of many computer systems is to retrieve data from a data store and display it for the user. After the user changes the data, the system stores the updates in the data store. Because the key flow of information is between the data store and the user interface, you might be inclined to tie these two pieces together to reduce the amount of coding and to improve application performance. However, this seemingly natural approach has some significant problems. One problem is that the user interface tends to change much more frequently than the data storage system. Another problem with coupling the data and user interface pieces is that business applications tend to incorporate business logic that goes far beyond data transmission.

The problem that this pattern is trying to solve is the following: How do you modularize the user interface functionality of a Web application so that you can easily modify the individual parts?

The following forces act on a system within this context and must be reconciled as you consider a solution to the problem:

User interface logic tends to change more frequently than business logic, especially in Web-based applications. For example, new user interface pages may be added, or existing page layouts may be shuffled around. After all, one of the advantages of a Web-based thin-client application is the fact that you can change the user interface at any time without having to redistribute the application. If presentation code and business logic are combined in a single object, you have to

modify an object containing business logic every time you change the user interface. This is likely to introduce errors and require the retesting of all business logic after every minimal user interface change.

In some cases, the application displays the same data in different ways. For example, when an analyst prefers a spreadsheet view of data whereas management prefers a pie chart of the same data. In some rich-client user interfaces, multiple views of the same data are shown at the same time. If the user changes data in one view, the system must update all other views of the data automatically.

Designing visually appealing and efficient HTML pages generally requires a different skill set than does developing complex business logic. Rarely does a person have both skill sets. Therefore, it is desirable to separate the development effort of these two parts.

User interface activity generally consists of two parts: presentation and update. The presentation part retrieves data from a data source and formats the data for display. When the user performs an action based on the data, the update part passes control back to the business logic to update the data.

In Web applications, a single page request combines the processing of the action associated with the link that the user selected with the rendering of the target page. In many cases, the target page may not be directly related to the action. For example, imagine a simple Web application that shows a list of items. The user returns to the main list page after either adding an item to the list or deleting an item from the list. Therefore, the application must render the same page (the list) after executing two quite different commands (adding or deleting)—all within the same HTTP request.

User interface code tends to be more device-dependent than business logic. If you want to migrate the application from a browser-based application to support personal digital assistants (PDAs) or Web-enabled cell phones, you must replace much of the user interface code, whereas the business logic may be unaffected. A

clean separation of these two parts accelerates the migration and minimizes the risk of introducing errors into the business logic.

Creating automated tests for user interfaces is generally more difficult and time consuming than for business logic. Therefore, reducing the amount of code that is directly tied to the user interface enhances the testability of the application.

Solution

The Model-View-Controller (MVC) pattern separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes [Burbeck92]:

- Model. The model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).
- View. The view manages the display of information.
- Controller. The controller interprets the mouse and keyboard inputs from the user, informing the model and/or the view to change as appropriate.

Figure 4.2. depicts the structural relationship between the three objects.

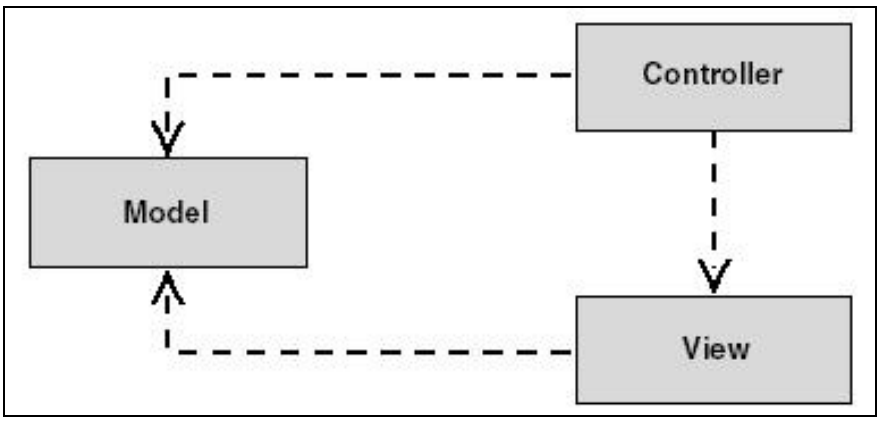

Figure 4.2 MVC Class Architecture

It is important to note that both the view and the controller depend on the model. However, the model depends on neither the view nor the controller. This is

one the key benefits of the separation. This separation allows the model to be built and tested independent of the visual presentation. The separation between view and controller is secondary in many rich-client applications, and, in fact, many user interface frameworks implement the roles as one object. In Web applications, on the other hand, the separation between view (the browser) and controller (the server-side components handling the HTTP request) is very well defined.

Model-View-Controller is a fundamental design pattern for the separation of user interface logic from business logic. Unfortunately, the popularity of the pattern has resulted in a number of faulty descriptions. In particular, the term "controller" has been used to mean different things in different contexts. Fortunately, the advent of Web applications has helped resolve some of the ambiguity because the separation between the view and the controller is so apparent.

Variations

In Application Programming in Smalltalk-80: How to use Model-View-Controller (MVC) [Burbeck92], Steve Burbeck describes two variations of MVC: a passive model and an active model. The passive model is employed when one controller manipulates the model exclusively. The controller modifies the model and then informs the view that the model has changed and should be refreshed (see Figure 3.3).

The model in this scenario is completely independent of the view and the controller, which means that there is no means for the model to report changes in its state. The HTTP protocol is an example of this. There is no simple way in the browser to get asynchronous updates from the server. The browser displays the view and responds to user input, but it does not detect changes in the data on the server. Only when the user explicitly requests a refresh is the server interrogated for changes.

The active model is used when the model changes state without the controller's involvement. This can happen when other sources are changing the data and the changes must be reflected in the views. Consider a stock-ticker display. You receive stock data from an external source and want to update the views (for example, a

ticker band and an alert window) when the stock data changes. Because only the model detects changes to its internal state when they occur, the model must notify the views to refresh the display.

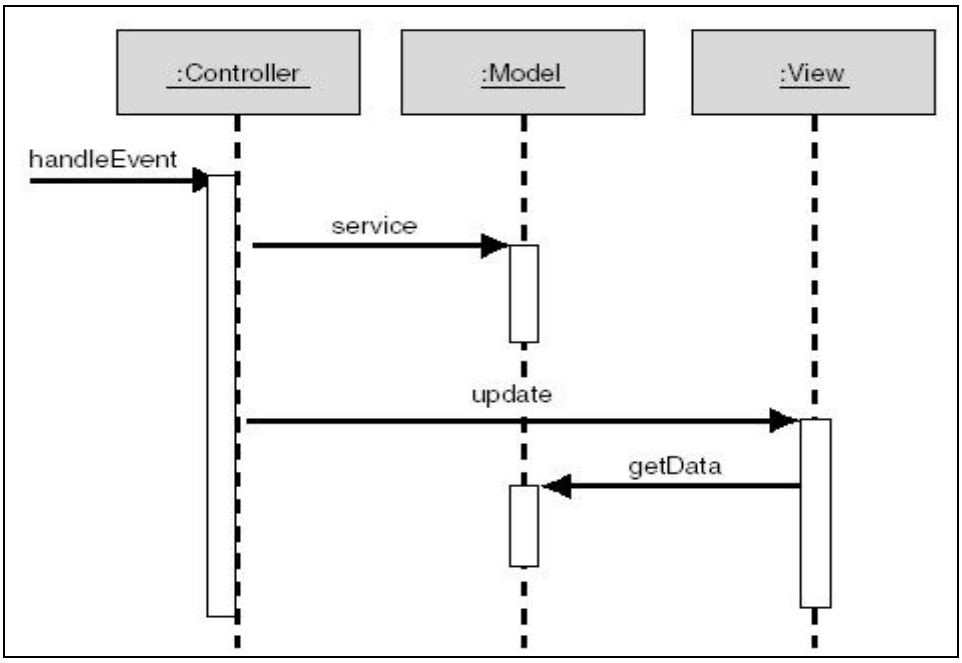

Figure 4.3 Behavior of the passive model

However, one of the motivations of using the MVC pattern is to make the model independent from of the views. If the model had to notify the views of changes, you would reintroduce the dependency you were looking to avoid. Fortunately, the Observer pattern [Gamma95] provides a mechanism to alert other objects of state changes without introducing dependencies on them. The individual views implement the Observer interface and register with the model. The model tracks the list of all observers that subscribe to changes. When a model changes, the model iterates through all registered observers and notifies them of the change. This approach is often called "publish-subscribe." The model never requires specific information about any views. In fact, in scenarios where the controller needs to be informed of model changes (for example, to enable or disable menu options), all the controller has to do is implement the Observer interface and subscribe to the model changes. In situations where there are many views, it makes sense to define multiple

subjects, each of which describes a specific type of model change. Each view can then subscribe only to types of changes that are relevant to the view.

CHAPTER 5

AN EXAMPLE AGENT APPLICATION RUNNING ON THE PROPOSED PLATFORM

This chapter focuses on the functional details of developed course management software namely Course ON-LINE and its intelligent agent add-on GAIA developed in the aforementioned agent development and execution platform.

5.1. An Overview of Course ON-LINE

Course ON-LINE is a member of e-university tools family which includes Campus ON-LINE [2], the course registration and student information system; Library ON-LINE [3], the university library automation system and CAMPUS ON-SMS [4], information distribution system over SMS. All of these tools are fully integrated with each other and share a common database.

Course ON-LINE uses the common database of the e-university tools for initially setting up of course homepages automatically as the new semester begins and generates a course syllabus for each course. Instructors can add custom information to the syllabus (i.e. course outline, course objectives, textbooks and references etc.), define grading scheme of the course and assign teaching assistants to the course. Instructors can give assignments, publish course materials, make announcements, add web based tools (i.e. web based simulations or animations related to the course), supply links to different web resources, announce grades of homework, projects or exams, or communicate with the students using forums or e-mail facilities.

Students can view course syllabus, view assignments and upload assignment files, reach to instructor's course materials, reach to web resources related to the course supplied by the instructor, view course announcements, grades of homeworks, projects or exams, view and participate to the forums and communicate with their instructors using e-mail.

Course ON-LINE supplies centralized management of all course homepages of a school and enables a uniform, easy to learn and use structure for each course homepage. It enforces Internet use in courses and generates a significant amount of publicly accessible course content.

With effective use of Course ON-LINE, the university gains the asset of structured and uniformly collected course content.

Course ON-LINE has four type of users; instructors, teaching assistants, students and guest users. Instructors and teaching assistants are the content suppliers and managers of their courses' homepages. A detailed explanation of each Course ON-LINE funciton and typical use of instructors facilities are demonstrated in Appendix A.

5.2. The GAIA Add-On

Gaia is the notification agent system of students running on the Course ON-LINE platform. Students use their own interfaces to start their own notification agents. To start a student agent, student selects the modules of the Course ON-LINE to be tracked by his/her own agent and gives directive to start the agent. (Figure 5.1)

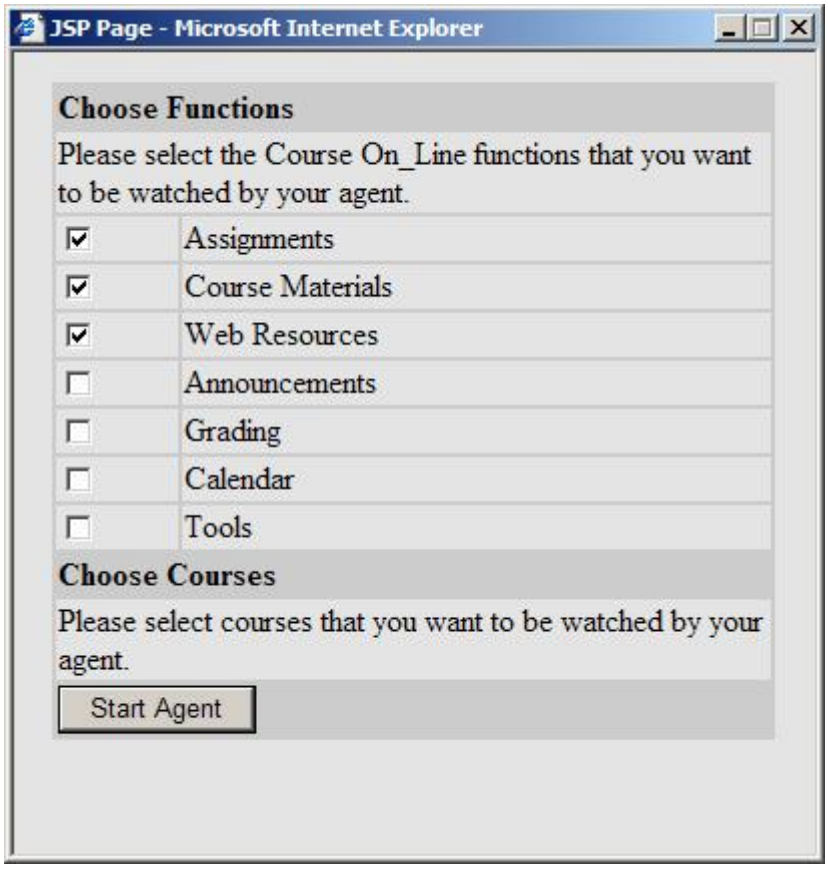

Figure 5.1 Student Agent Configuration Interface

5.2.1. How GAIA works

GAIA is the creator and living environment of each student's notification agent. There are three modules that are running the environment. First module is the main constructor of the agent platform called Gaia Agent Startup (GAS). This module starts when the Course ON-LINE server starts up. It is a console application which is responsible for starting the JADE platform and initiating the preconfigured agents to be created and started to monitor.

Second module is the agent environment developed by Java and running in the JADE agent platform. The mother agent 'GAIA' and child agents 'Titans' live in this environment.

Third module is the user interface module that is accessible by students from Course ON-LINE interface and enables students to turn on and off their own notification agents (Titans).

If it is the first time that GAS is running, it initially creates a main agent called GAIA. The only purpose of this mother agent is to create child agents (Titans) when a new agent request comes from a student. Actually GAIA is the name of this mother agent. The GAIA agent monitors the Course On-Line database for every 5 seconds and looks for a new agent request from a student. If finds a new one, creates a titan and goes asleep for next turn. Each titan is born with an array of parameters that holding the Course On-Line functions to be monitored.

If for any reason the server is stopped or crashed, the previously running agents' information is kept in the database. So if server is started for second time, the GAS utility checks for this situation and recreates all titans which were previously running. This ensures the continuity of the agent system.

When the GAS starts the JADE platform and creates GAIA agent, it starts to monitor Course ON-LINE database for new agents every 5 seconds. If a new agent request is found in the database, GAIA creates a new Titan and sleeps. The first thing that a Titan does is to learn which functions of Course On-Line it should monitor on behalf of the student. This information is gathered by the Gaia agent and sent to the newly born Titan as startup parameter. After learning which functions it should monitor, titan starts to monitor the relevant tables in the Course ON-LINE database and sends an e-mail notification if it finds out a new entry in those tables. The titan does this operation continually until the owner of the agent stops it.

Figure 5.2 shows the lifecycle of the monitoring agents.

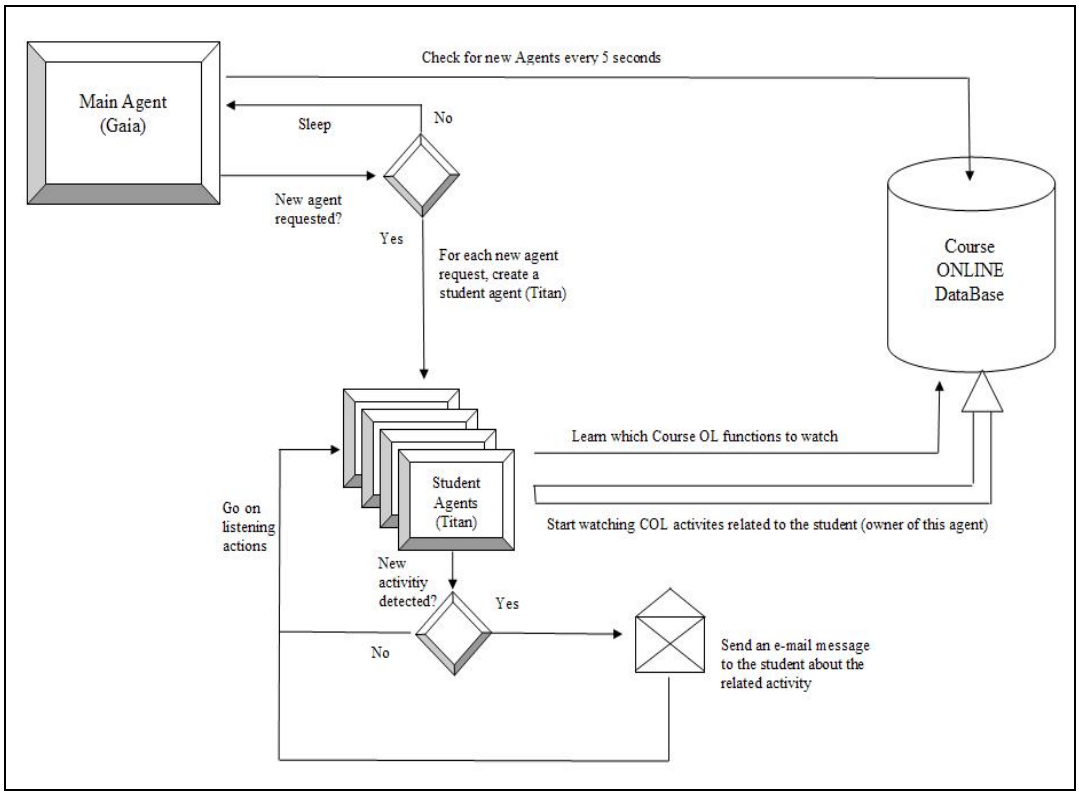

Figure 5.2 Student Agents Lifecycle

5.3. Integrating Course ON-LINE With Other University Information Systems

Course ON-LINE is a part of other university information systems available currently used in Işık University. These are Campus ON-LINE; student registration and information system, Library ON-LINE; library management system, Campus ON-SMS; SMS based information distribution system, Pay ON-LINE; electronic payment system and Access ON-LINE, electronic access control system. Course ON-LINE, in this sense, has conceptual and physical connections with these systems. In this section, information about the integration is delivered.

The integration is done in two different layers: Integration at the data layer, and integration at the business layer. These will be described in detail.

5.3.1. University Information Systems

Campus ON-LINE

Campus ON-LINE is the student information and online registration system of IŞIK University. Students use this system when first registering to the university and when registering to courses at the beginning of each semester. Instructors use the system to see class lists during the semester, submit letter grades at the end of each semester and advise online to students during the registration periods. There are also some functions defined for some administrative units.

There are more that 2000 users of the system, which are, grouped into more that 10 different user groups. As development technology, ASP is used with MS SQL Server 2000 as database management system. Components are developed in Visual Basic 6.0.

Library ON-LINE

Library On-Line is the library management software currently used at Işık University. Library ON-LINE is a very efficient and powerful library automation system. Currently it has four key facilities are functional. These are Cataloging, Circulation, WebOPAC and Administration modules.

The development is done in J2EE and running on the Sun ONE Application Server platform using Microsoft SQL Server 200 as the DBMS.

Campus ON-SMS

Campus ON-SMS is an SMS based information distribution service. The service currently, delivers course final notes or announcements from the university to the students who are registered to the system.

Campus ON-SMS is based on the Microsoft SQL Server 2000 and transact-sql language.

Pay ON-LINE

Pay ON-LINE is an electronic payment system. The system is in use in IŞIK University campuses. The system allows payment in campuses to be done through an

electronic proximity card and holds the student information in this card. The system has reports for the accounting department and other departments of interest.

Pay ON-LINE is developed in Microsoft .NET platform and uses Microsoft SQL Server as a DBMS.

5.3.2. Integration at Data Layer

The information system above all share a common user database. The users of the system are students, instructors (part and full time), alumni, administrative staff, and sub-contractors of the university that access to the university. There are also guest users in the system. The user database is therefore designed to be common.

The common user database is responsible only from storing the student (or administrative) id numbers, passwords, and names. Other types of information are left out, because each system needs distinct data. For example, Campus ON-LINE needs to distinguish between the grades of the students, while Library ON-LINE needs to know whether he/she is undergraduate or graduate. Storing of user types are also left to the individual systems, as the level of granularity changes from one system to other.

This common database scheme has the following advantages:

- Redundant data is kept minimal. When there is not a common database, all system should keep track of a common set of information on its own separately. This means that the same information would exist in many databases.
- Simplifies record updates. Keeping the same data across multiple databases delivers the problem of updating this data. For example, when student number changes or a student graduates, the related data on all of these databases should be updated. Some trigger mechanisms might be deployed, but this is problematic as well, as these trigger programs at the database level, makes the database design more complex than it should be and makes the system platform

dependent as these triggers should be re-coded if the database system should change. Likewise, every time a user changes a password, the change should be propagated though all databases.

- Enables some common software modules to be used by all systems. A good example is the common authentication system which is described in detail in the next chapter. With the module all the authentication is done through a common application, which is a web service.
- Reduces the need for storage. Minimizing the existence of redundant data reduces the need for physical storage and helps increase the overall performance.
- Simplifies operational processes. This is an important issue when the systems are in interest of various university departments. For example, undergraduate students' information is under control of the student registrar department, while the student's tuition information is under control of the accounting department. The common database model allows the processes to be separated. A change in student information is done only through a single database, and does not need to propagate.
- Reduces complexity of the database. Eliminating redundant data reduces the database complexity and dependencies across the databases.

There is only one table in the common database, which is the user table. It has two attributes; *userid* and *password*. The *userid* attribute is the student id or the administrative id of the user. *Password* attribute is the SHA1 [27] hash of the password. It is a fixed 40 character column. The SHA1 hash of a string is a 40 character hexadecimal number.

When a message of any length < 2^64 bits is input, the SHA-1 produces a 160-bit output called a message digest. The message digest can then, for example, be input to a signature algorithm which generates or verifies the signature for the message. Signing the message digest rather than the message often improves the efficiency of the process because the message digest is usually much smaller in size than the message. The same hash algorithm must be used by the verifier of a digital signature as was used by the creator of the digital signature. Any change to the

message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify. The SHA-1 is called secure because it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest. Any change to a message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify.

The common database as seen above has a very simple design but is sufficient and cost effective for the university information system.

5.3.3. Integration at the Business Layer

The university information systems used does not share many functionality except the system operation which are handled by the development platforms. One common function implemented is the authentication.

All of the university information systems share a common user base, and as described above a common database is designed for this user base. For this user database, a common authentication software module is implemented and used across all systems.

The problem that arises when implementing a common software module is that all of the systems in the university information system are developed in different platforms. Some are in Microsoft Active Server Pages (Visual Basic 6.0), some are in Microsoft .NET, and some are developed in Java platform. Although there are many ways of sharing common modules across these platforms, most of them are inefficient and costly. Some methods are listed below:

- Using a C module: Windows native Dynamic Link Libraries written in C are accessible by both Visual Basic 6.p and the .NET platform. Java can also access these components through the JNI (Java Native Interface) [28]. The JNI allows Java code that runs within a Java Virtual Machine (VM) to operate with applications and libraries written in other languages, such as C, C++, and

assembly. In addition, the Invocation API allows you to embed the Java Virtual Machine into your native applications. Using this method has many drawbacks. First the performance is limited. The authentication module is busy software, as it handles the entire authentication from all the university information systems. The other problem is coding and accessing these modules is programmatically hard to implement.

- Using Middleware Architecture: Middleware architectures like the CORBA can be used, but again implementing these architectures are hard.
- Using a XML web service: The most appropriate solution is using this method. Java, .NET and the Visual Basic 6.0 has methods or libraries that simplify accessing XML web services.

From the above choices, we took taking XML web services into account. A software module was implemented complying with the web services standard in the Java programming language. The web service uses SOAP [29] over HTTP to communicate with the clients. The clients in this case are the application software like the Library ON-LINE, Campus ON-LINE, Course ON-LINE, and Pay ON-LINE. These applications software send the username / password pair to the web service in XML. The web service calculates the SHA1 [27] hash of the password and authenticates it against the pair stored in the user database. The web service responds with ‘true’ or ‘false’ Boolean value.

CHAPTER 6

EVALUATION

Course ON-LINE is in use at Isik University already. For the fall semester of the 2003-2004 academic year, a total 178 course homepages were set by the system, involving 190 instructors and 1986 students. Table 6.1. summarizes the comparison of active usage of Course ON-LINE in two consequent semesters.

Table 6.1 Comparison of active usage of Course ON-LINE in two semesters

	2002-2003 Spring	**2003-2004 Fall**
Total number of courses opened in the semester	129	178
Total number of course sections opened in the semester	329	392
Number of courses having actively used web page in Course ON-LINE	22	173

GAIA is the notification agent of students using Course ON-LINE. There are three modules that are running the GAIA agent. First module is the main constructor of the agent platform. Second module is the agent environment developed by Java and running in the JADE agent platform. The mother agent 'GAIA' and child agents 'Titans' live in this environment. Third module is the user interface module that is accessible by students from Course ON-LINE interface and enables students to turn on and off their own notification agents.

CHAPTER 7

CONCLUSION AND RECOMMENDATIONS FOR FUTURE WORK

In this thesis, we reported an experience on constructing a software agent platform for development and implementation of software agent systems running with integrated applications which are developed and running under different technologies. The proposed platform consists of an agent development framework namely JADE, a common database infrastructure serving to many different applications and the applications infrastructure running on different platforms.

The example application running on the proposed platform consists of an integrated course management software, namely Course ON-LINE, and an agent application running as a notification agent of Course ON-LINE namely GAIA.

Course ON-LINE offers a uniform and centralized management of all course homepages of a school and encourages Internet use in courses. It generates a significant amount of publicly accessible course content which is a very important asset for a university. With advanced features using intelligent agents, the functionality and user friendliness is achieved.

The example agent application does not include many advantages that are brought by agent software like pro-activeness, social abilities, and mobility. The example is aiming to demonstrate the use of proposed platform in conjunction with running integrated applications.

Improvement of the agent based systems by introducing more intelligence, autonomy and mobility to agents running on the proposed platform would be next steps of future work. As demonstrated in second chapter of this work, introducing the core functionality and usability of software agents highly depends on their mobility.

From this point of view, developing mobile software agent applications in this environment would enhance the usability of the system.

Other research topics on the field might be performance issues and enhancements of using the proposed platform, or scalability problems that may arise and security problems.

Obviously, using an existing open source environment for agent software development might lead the developers and system architects to limit themselves on the existing platform. A typical research area therefore would be to extend the capabilities of the existing agent platforms or replacing the agent development platforms with other alternative either by developing a new one from scratch or another existing alternative.

REFERENCES

[1] A.Jafari, "*Conceptualizing Intelligent Agents for Teaching and Learning*" Educase Quarterly, No 3, 2002 p 28-33

[2] Campus ON-LINE Web Site, http://campus.isikun.edu.tr

[3] Library ON-LINE Web Site, http://library.isikun.edu.tr

[4] Campus ON-SMS Web Site, http://irdc.isikun.edu.tr/projects/campusonsms

[5] Gilbert, D.; Aparicio, M.; Atkinson, B.; Brady, S.; Ciccarino, J.; Grosof, B.; O'Connor, P.; Osisek, D.; Pritko, S.; Spagna R. and Wilson, L., "*IBM Intelligent Agent Strateg"y*, IBM Corporation, (1995).

[6] Kostakos V., Taraschi C., "*Agents*", May 7, 2001

[7] FIPA Web Site, http://www.fipa.org

[8] Iglesias, C.A. and González, J.C. "*A Survey of Agent-Oriented Methodologies*" In Proceedings of the 5th International Workshop on Agent Theories, Architectures and Languages (ATAL'98), LNAI n1555 - Springer Verlag, Paris, France, July 1998, pp:317-330.

[9] Finin, T.; McKay, D. and McEntire, R., "*KQML as an Agent Communication Language*". In: Proc. 3rd Int. Conf. Information and Knowledge Management, (Adam, N. R. ed.), ACM Press, 1994.

[10] The Agent Society Web Site, http://www.agent.org/

[11] W. Brenner, R. Zarnekow and H. Wittig. "*Intelligent Software Agents: Foundations and Applications*." Springer, 1998

[12] Chess, D.;Harrison, C. and Kershenbaum, A., "*Mobile Agents: Are They a Good Idea?",* 1997, pp. 25-47

[13] Birman, K. P. and van Renesse, R., "*Reliable Distributed Computing with the ISIS Toolkit.*" IEEE Computer Society Press, 1994.

[14] IBM Corp., "*Messaging and Queueing Technical Reference"*, SC33-0850, 1993.

[15] McDermott, Dominic, "*Is the Internet the Way For Teaching To Go Forward?*", The Culture of Publishing, June 2001, Oxford Brookes University Press

[16] Gilbert, Alan "*The Virtual University*", The Virtual University Symposium, 21-22 November 1996. The University of Melbourne, Melbourne.

[17] Open CourseWare Web Site, http://ocw.mit.edu

[18] EduTools Web Site, www.edutools.info

[19] SILVEIRA, Ricardo Azambuja, VICARI, Rosa Maria. "*JADE - Java Agents for Distance Education framework".* DEC 2001, 2001, Austin. DEC 2001. CD-rom, 2001

[20] F. Bellifemine, G. Caire, A. Poggi, G. Rimassa, "*Jade, A Whitepaper*", exp - Volume 3 - n. 3 - September 2003

[21] Michael Berger, Steffen Rusitschka, Dmitri Toropov, Michael Watzke, Marc Schlichte, "*Porting Agents to Small Mobile Devices –The Development of the Lightweight Extensible Agent Platform*", exp - Volume 3 - n. 3 - September 2003

[22] BlueJADE Project Web Site, http://sourceforge.net/projects/bluejade

[23] LGPL license, http://www.opensource.org/licenses/lgpl-license.php

[24] JADE Web Site, http://jade.tilab.com/

[25] Grundgeiger D., "*Programming VisualBasic .NET*", O'Reilly Publication, January 2002, p. 13-14

[26] Petroutsos E., "*Mastering Visual Basic .NET*", Sybex , 2002

[27] Eastlake, D., Jones, P., "*US Secure Hash Algorithm 1 (SHA1)*" Network Working Group, RFC: 3174, 2001

[28] Altendorf E., Hohman M., Zabicki R., "*Using J2EE on a Large, Web-Based Project*", IEEE Software, pp.81-89, March 2002

[29] Mitra N., "*SOAP Version 1.2 Part 0: Primer*", W3C Recommendation, 2003

[30] Stallings, W., "*Cryptograhy and Network Security*" pp. 362, 2001

[31] Curbera F., et al. "*Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI*" IEEE Internet Computing, pp. 86-93, March, 2002.

[32] S. Hayes, "*An Introduction to extreme Programming*" Second International Conference on eXtreme Programming and Agile Processes in Software Engineering, Sardinia, 2001

[33] W.C. Wake, "*The XP series: Extreme Programming Explored"*, Addison-Wesley, NJ, 2002

[34] Kelley, C., "*The Waterfall Model*" http://www.jsc.nasa.gov/bu2/PCEHHTML/pceh.htm, 2000.

[35] McCormick M., "*Programming Extremism*" Communications of the ACM, Volume 44, No. 6, 2001

[36] Yıldız M., "*New Practices For Extreme Programming Applied In Campus On-Line, A Large Web Based Application Development Project"*, MS Thesis, Isik University, January 2003

[37] Cox, Brad J., "*Object-Oriented Programming, An Evolutionary Approach*", Addison Wesley, NJ, 2001

[38] Martin J., Odell, J. J., "*Object-Oriented Analysis and Design*", Prentice-Hall, Englewood Cliffs, NJ. Pp241, 1998

[39] Booch, G., "*Object-Oriented Analysis And Design With Applications*", 2nd Ed. Benjamin Cummings. ISBN 0-8053-5340-2. pp.83, 1998

[40] Rumbaugh J., et al., "*Object-Oriented Modeling and Design*", Prentice Hall, NJ, 1997

[41] Shlaer S., Mellor S. J., "*Object-Oriented Systems Analysis: Modeling the World in Data*", Pp14, 1996

[42] Jacobson I., et al. "*Object-Oriented Software Engineering - A Use Case Driven Approach*", ACM Press/Addison Wesley, 1994

[43] Trowbridge D., Mancini D., Quick D., Hohpe G., Newkirk J., Lavigne D., "*Enterprise Solution Patterns Using Microsoft .NET*" Version 2.0, Microsoft Press, December 2002

APPENDIX A

DETAILS OF COURSE ON-LINE FUNCTIONS

A.1. Course ON-LINE Functions

Figure A.1. shows the main page of Course ON-LINE. Guest users may reach to the course homepages using the "List of Courses" menu on the left side. Other users reaches to their own interfaces by logging in to the system. All users can conduct search on course homepages using the "Search" facility.

After instructor logs in, the left side menu customizes for that user and lists the courses of that instructor. By clicking to the link of a specific course, instructor reaches his/her own course's homepage.

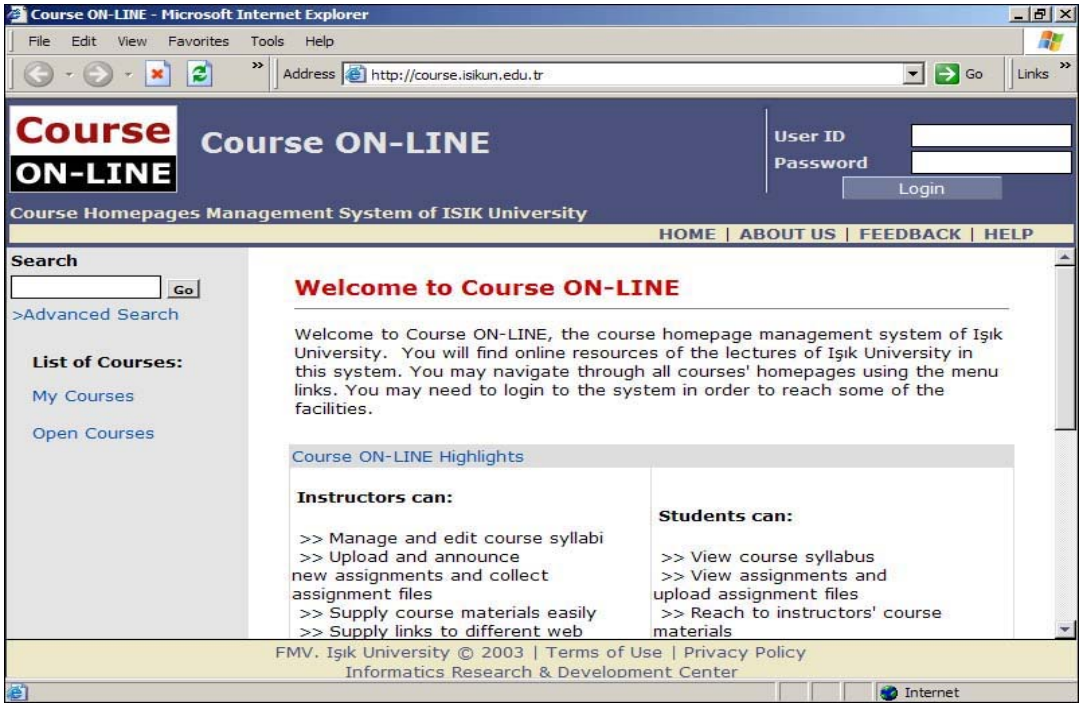

Figure A.1 Course ON-LINE main page

A.1.1. Course Syllabus

The first page of each course is the “Syllabus” page. Instructors can view and edit syllabus information on this page. Figure A.2. shows the syllabus page.

Some of the items in the syllabus come from Campus ON-LINE, so they are read only (eg. Prerequisite Courses, Corequisite Courses). You can define textbook, reference book of the course, assign teaching assistants to the course. You can define custom course requirement using Other Requirements field. You can also define course outline, course objectives and grading policy items.

Instructors can assign teaching assistants for the course and edit access privileges for that teaching assistant to the specific Course ON-LINE functions for that course. After the instructor grants access, that assistant may reach to the instructor’s user interfaces by logging in to the system with his/her own username and password.

If instructor has more than one sections of a particular course, he/she will see an extra field named “Apply to Sections” at the bottom of each page. This field gives him/her the opportunity to apply the syllabus to multiple sections of this course.

A.1.2. Assignments

Instructors can give assignments by uploading assignment files to the system, and download and view students’ answers to the assignments. There are some advanced features of this assignment facility. For example instructor can define a late submission penalty for an assignment and system automatically calculates the penalized grade for assignments which are submitted after assignment due date.

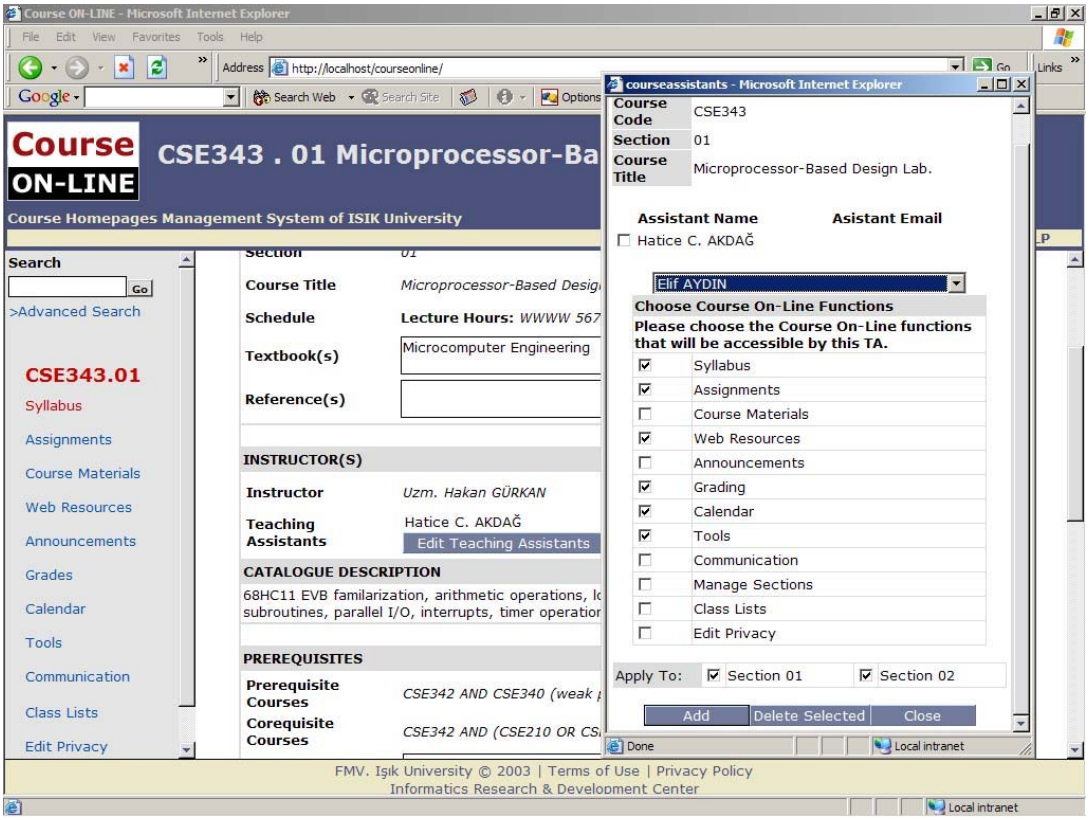

Figure A.2 Syllabus Page

To add a new assignment, instructors are asked to supply some information in the Add New Assignment form. Each assignment defined here should correspond to an item in the Grading Scheme that was previously defined in the *Syllabus Page*. Each grading scheme item can be found in the "*Apply To*" column of the "*Add New Assignment*" form. Students will see instructor's assignments from the course page and will upload their homework files. To define valid file types that the students may upload, instructor can use "*Valid File Types*" field. To allow students upload any type of files, 'all' must be chosen. To allow students to upload some file types, the file extensions separating with commas should be entered. (eg. doc,xls).

Publishing an assignment is realized by following these steps:

• Before defining assignments, he/she should define grading policy scheme.

• Instructor must define a name, upload a file, and select a due date for each assignment.

• To apply a penalty to the assignment he/she must check Apply Rule checkbox.

• To define valid file types for an assignment he/she should put commas between file types (e.g. doc,xls).

• If instructor deletes a homework item, all information related to this homework item will be lost (e.g. grades).

• If he/she is an instructor of more than one section of this course, instructor may add the same assignment for other sections of this course by using Apply to field as well.

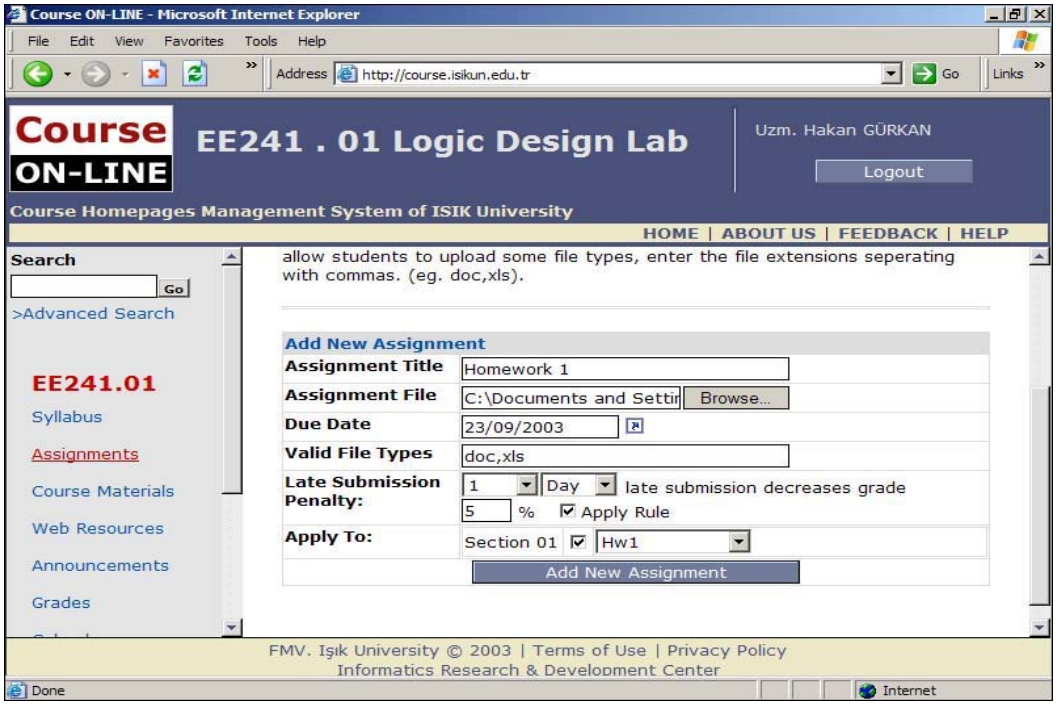

Figure A.3 Assignments interface of the instructor

A.1.3. Course Materials

Instructor can supply course materials by uploading files to the system, or using the “Course Materials” page.

Instructor may upload course materials such as handouts, slides, etc. using this page. Instructor can also delete the uploaded course materials by using the checkbox near the material name and clicking the *“Delete Selected”* button. If an instructor is tutoring more than one section of this course, he/she may add the same material for other sections of this course by using *“Apply to”* field as well. Figure A.4. shows a snapshot of Course Materials Page.

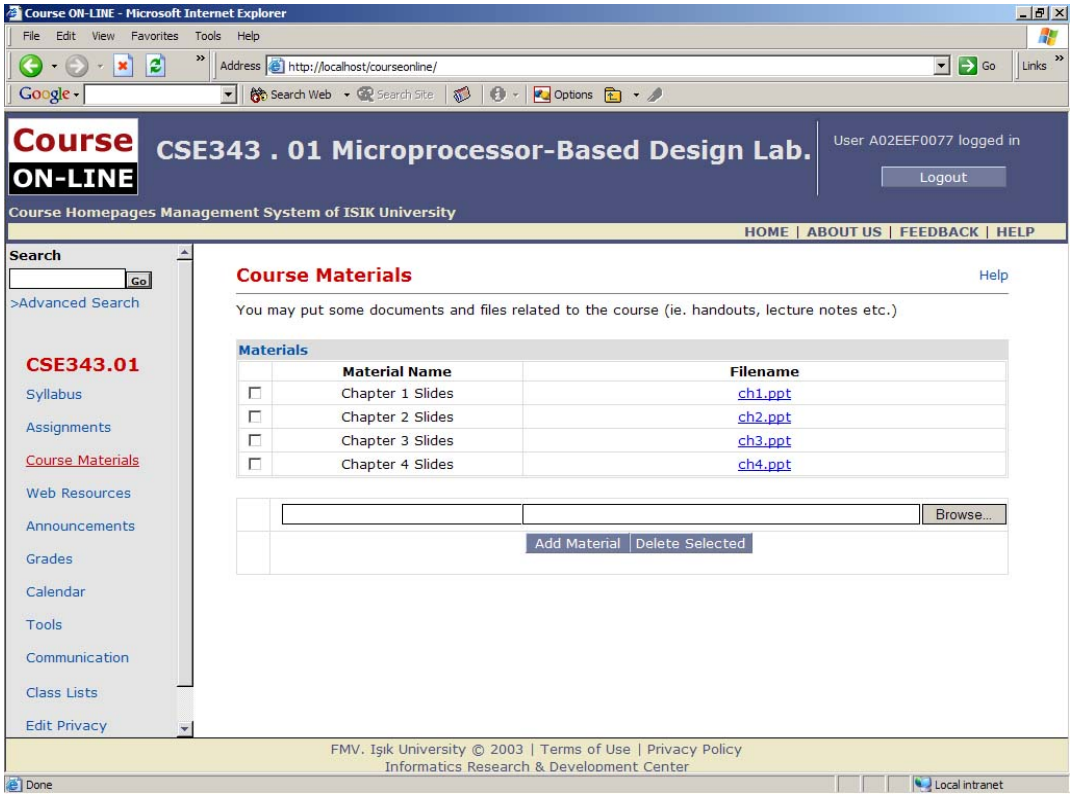

Figure A.4 Course Materials Page

A.1.4. Web Resources

Instructor can supply links to different web resources using the "Web Resources" page. He/she may add web page links related to the course by giving it a title, explanation and url. He/she can also delete the web resources defined before by using the checkbox near the title of the resource and clicking the *"Delete Selected"* button. Figure 5.5. shows a snapshot of the Web Resources Page.

A.1.5. Annoucements

Instructor may add announcements related to his/her course by defining its subject and content. He/she can also delete the announcements defined before by using the checkbox near the title of the announcements and clicking the *"Delete Selected"* button. If instructor tutors more than one section of this course, he/she may add the same announcement for other sections of this course by using *"Apply to"* field as well. Figure A.6. shows a snapshot of the Announcements Page.

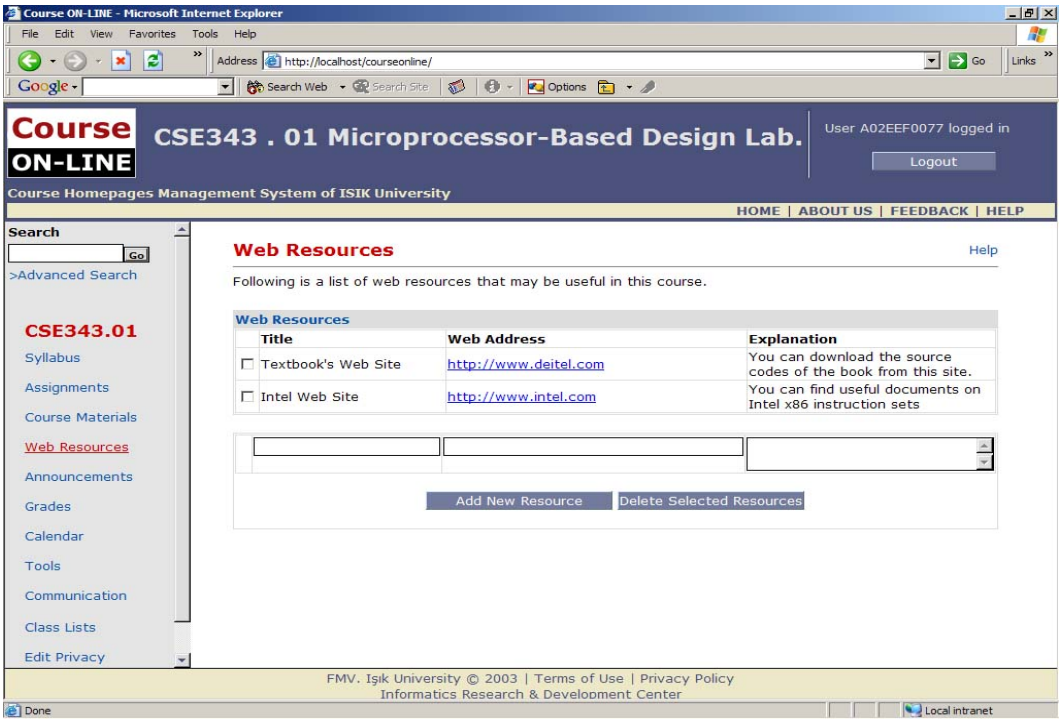

Figure A.5 Web Resources Page

A.1.6. Grading

Grading is one of the most important tools of Course ON-LINE. Instructor can grade each work done in the course using this tool. After grading assignments, quizzes, exams or attendances, overall grades are automatically calculated and reported to the instructor; with various statistical information, like maximum and minimum grades, averages and standard deviations.

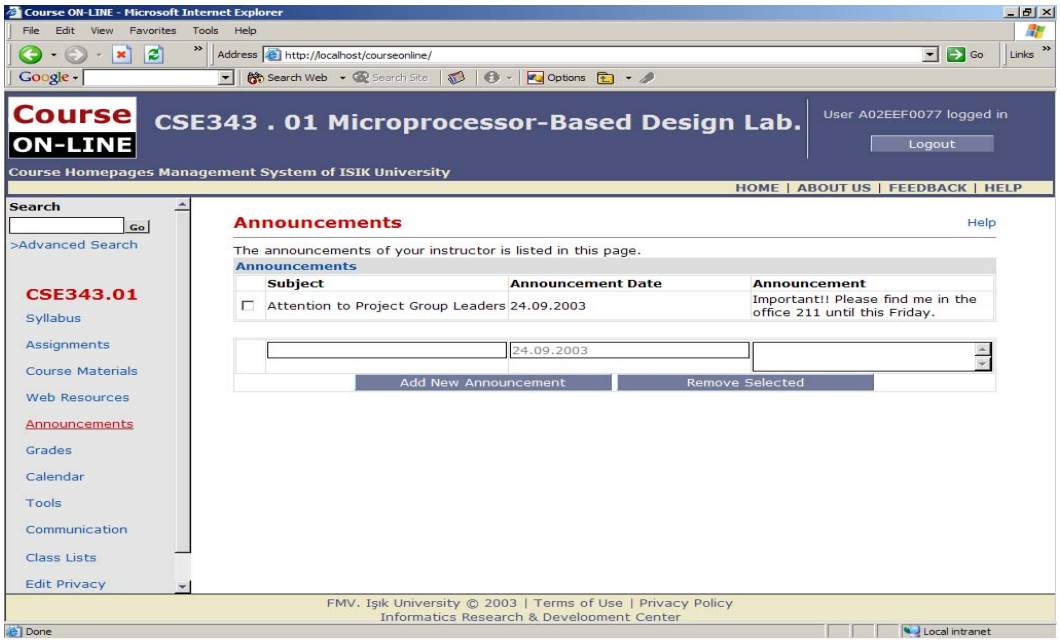

Figure A.6 Announcements Page

Instructors enter grades for each item in their course and calculate overall grades by using this interface. The grading policy items for the course will be listed under the *"Item List"* part. Figure.A.7. shows the first page of the Grading Tool.

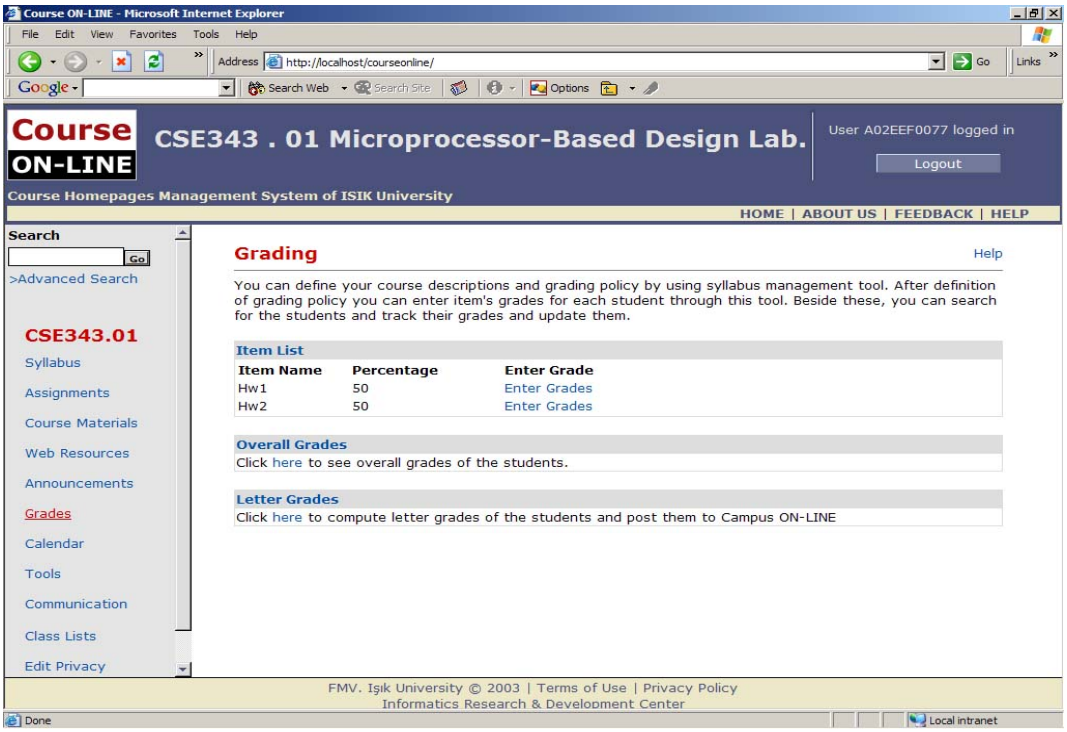

Figure A.7 First Page of the Grading Tool

Using Enter Grades link, instructor reaches to the specific page of that grading item to give students' grades (Figure A.8.). If grades related to this grading policy item are entered before, they are listed and all changes will be overwritten.

Students can not see their grades until instructor clicks the submit button. If instructor wants to apply a penalty to the student grade, he/she must check the checkbox under the Apply Penalty heading of the table.

The other important parameter that can be defined in the *Syllabus Page* is the "Grading Scheme". In the grading scheme, instructor supplies the grading items and their percentages. This scheme is required for giving assignment and grade calculations.

To define a grading scheme, instructor should repeat supplying an item name and give a percentage for that item. If he/she wishes to give a weight to course

attendance, he/she should also enter a grading item with the name “Attendance” and mark the attendance checkbox.

Using Overall Grades link instructor can view the overall grades of the students (the weights of the item are applied to the grade) for this course. Instructor can view overall grades of the class, maximum, minimum and average grades of each grading policy item. Instructor can also sort the grades according to each grading policy item and total in both descending and ascending order. (Figure A.9)

In the Letter Grades part, instructor may calculate and submit/save letter grades of the course. Instructor must define the grading policy of the course in the *Syllabus Page*, otherwise he/she will not be able to enter or calculate grades. If letter grades are calculated before, they are listed and after recalculation they are overwritten. Instructor can define lower limits for each letter grade and the overall grades are automatically converted to letter grades. Instructor can make further arrangements on the pre-calculated letter grades manually. (Figure A.10)

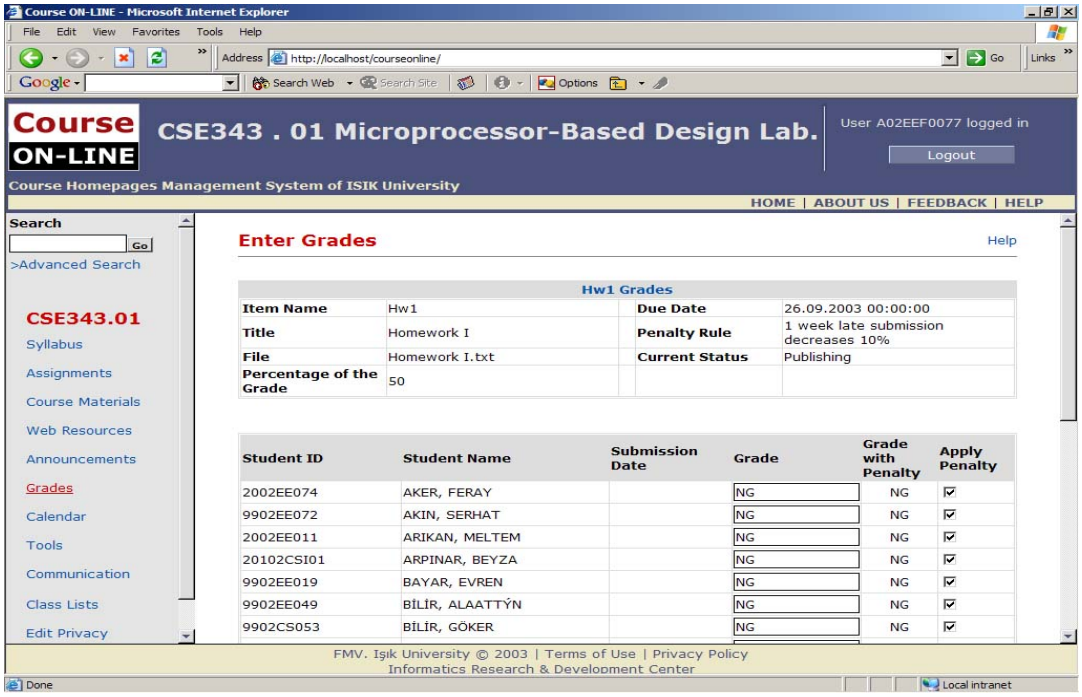

Figure A.8 Entering Grades for Each Grading Item

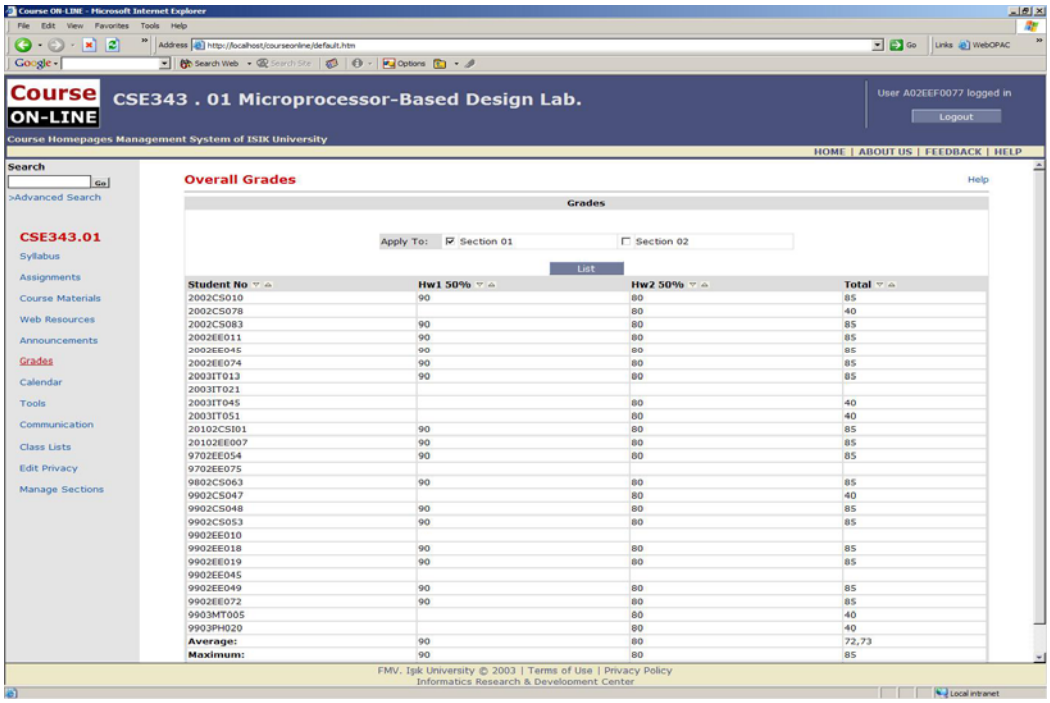

Figure A.9 Viewing Overall Grades

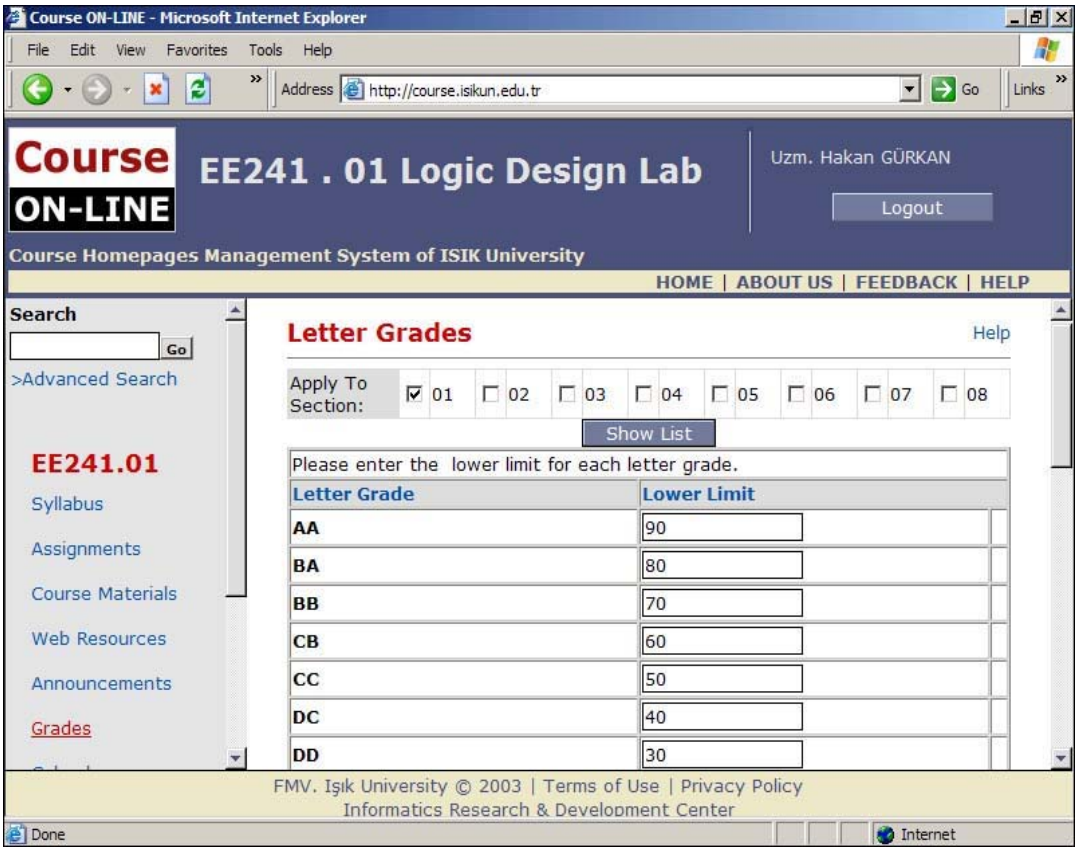

Figure A.10 Letter Grades Conversion Tool

Using the “Letter Grades” page, instructor can convert the overall grades to letter grades and submit the final grades to Campus ON-LINE. These grades will also be published in the students’ interfaces automatically.

A.1.7. Calendar

Instructor may add new events to the calendar or delete events from the calendar. He/she can also list events according to time period in ascending or descending order. To publsh a new calendar event, instructor must enter the Event Title and Event Date fields. To delete and event instructor should check the checkbox near the Event Title before clicking the Delete Selected button. If instructor is a tutor of more than one section of this course, he/she may add the same material for other sections of this course by using Apply to field as well. A snapshot of course calendar is shown in Figure A.11.

A.1.8. Tools

Instructors can also supply custom web based tools related to the course using the “Tools” page. These tools may be animations or simulations running on the web (like Java applets or Flash animations). This function is added to the Course ON-LINE to give flexibility to the instructor to use any kind of custom web based course material in the course. (Figure A.12.)

A.1.9. Communication Forums

Communication is the one of the most important functions of a course management software. It is very crucial for an instructor to communicate with the students and other teaching staff during the course. Course ON-LINE supplies two kinds of communication channels. One is the course forums. Instructors can open and manage forum threads about their courses.

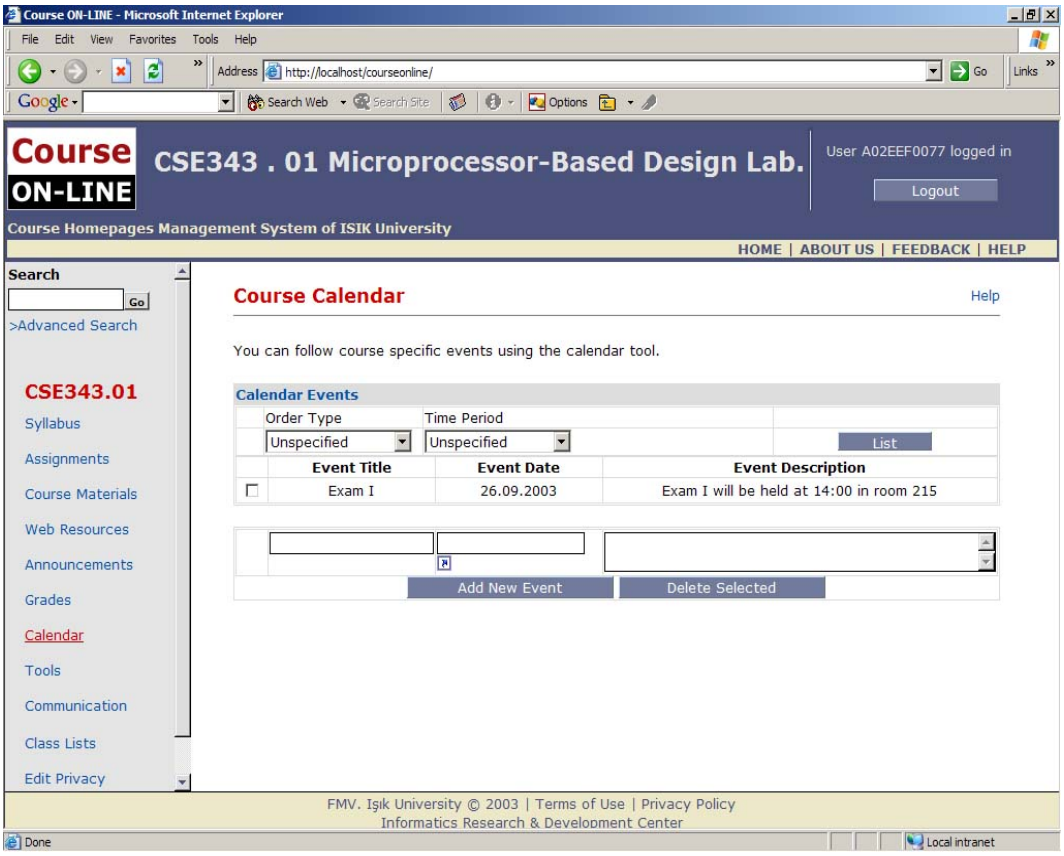

Figure A.11 Course Calendar

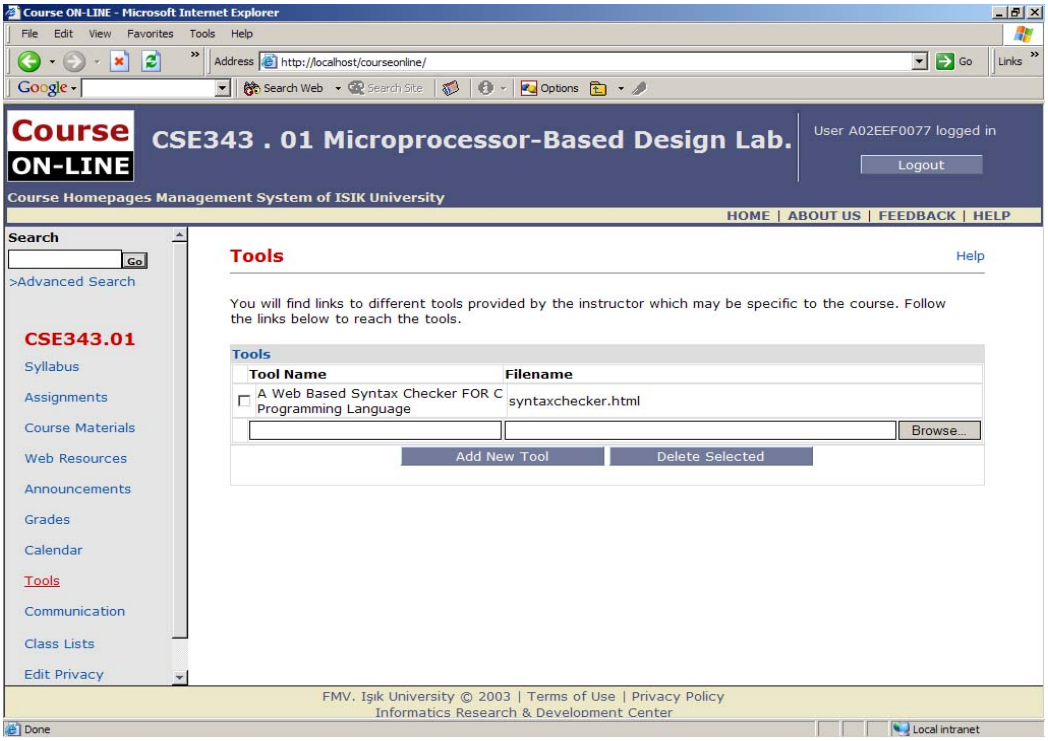

Figure A.12 Tools Page

After opening a new forum thread, students can view and reply to the messages. Instructor can set a rule if he/she wants to check students' messages before they are published in the students' forum pages. Or he/she may allow students to send messages instantly without any pre-activation. To adjust these rules instructor uses the "Forum Rules" tab. Instructor may also view and activate/deactivate existing messages by clicking on the relevant message. Instructor may open the same forum thread on multiple sections of this course by using *"Apply to"* checks. Figure A.13. shows a snapshot of Forums pages.

A.1.10. Sending Batch E-Mails

Sending batch E-Mails is another communication channel supplied within Course ON-LINE. Instructor can send batch e-mails to his/her teaching assistants and students using this facility. Instructor can also define permanent groups for sending e-mails. By defining those groups instructor can send batch e-mails to the mail groups more easily. (Figure A.14.)

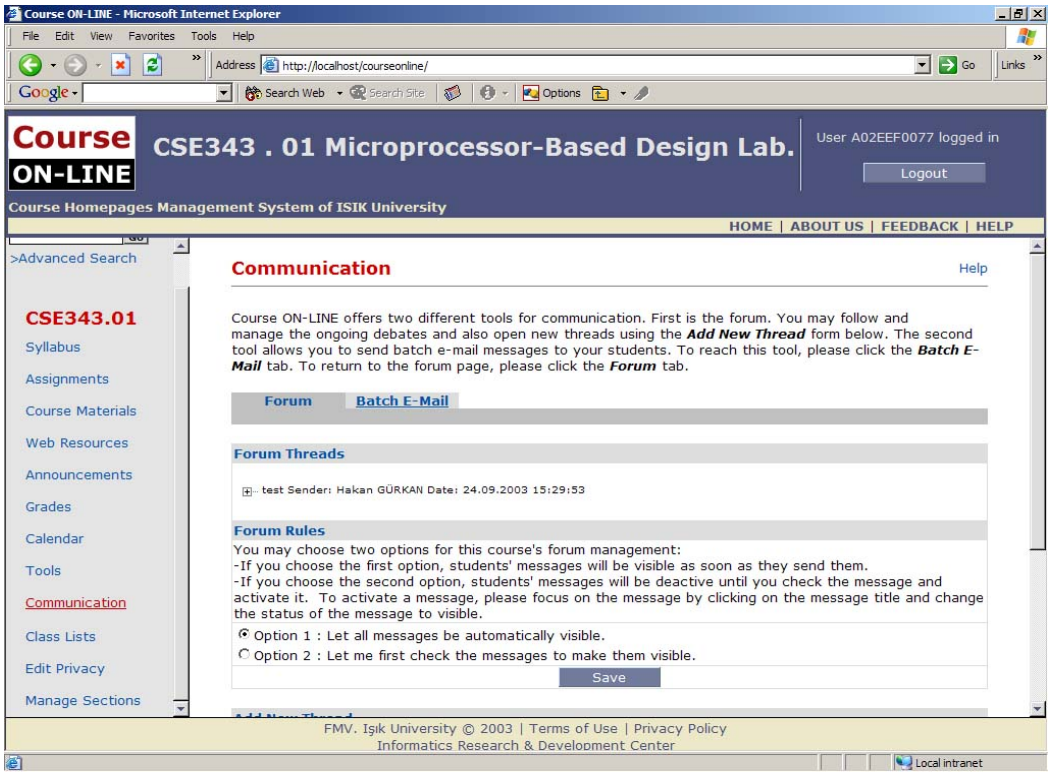

Figure A.13 Communication - Forums

A.1.11. Class / Attendance Lists

Attendance handling is another facility of Course ON-LINE. Instructors can enter the attendance information using "Class/Attendance Lists" page and this information will be used in grade calculation if attendance has a grading weight defined in the grading scheme.

A.1.12. Advanced Features

Course ON-LINE offers advanced features on management of the course homepages. Coordination of course sections is one of these advanced features. If an instructor owns several sections of a course, he can apply his actions to multiple sections of a course using the "Apply to Sections" checkboxes. (Figure A.15)

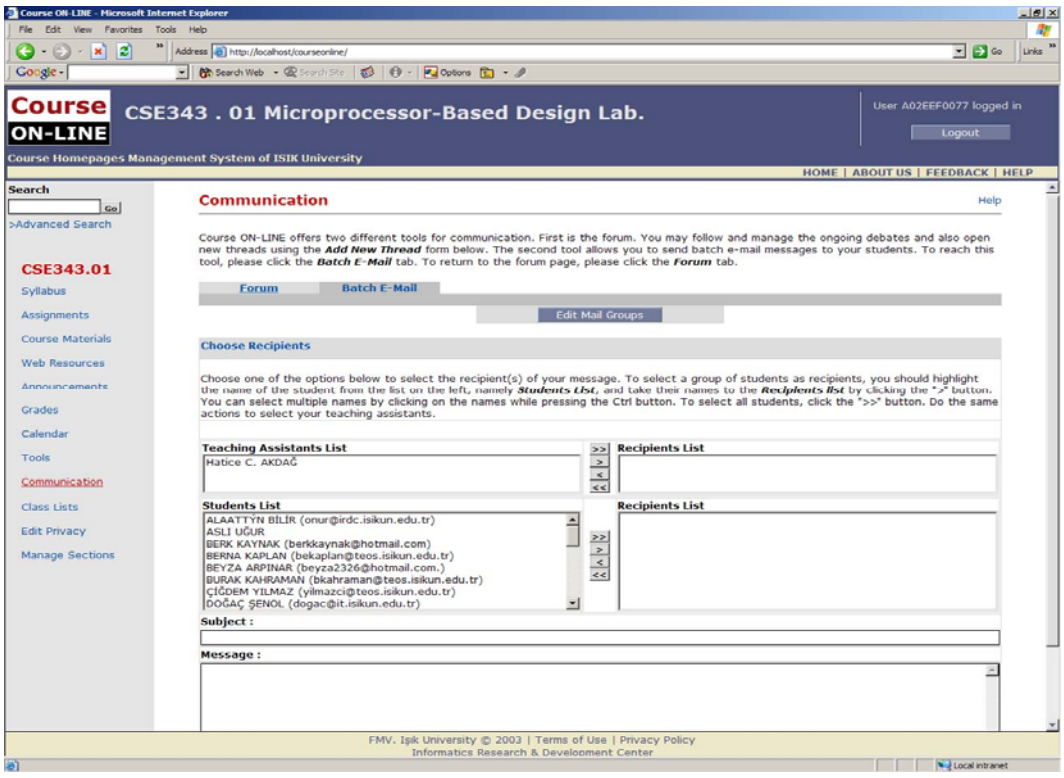

Figure A.14 Communication – Batch E-Mail

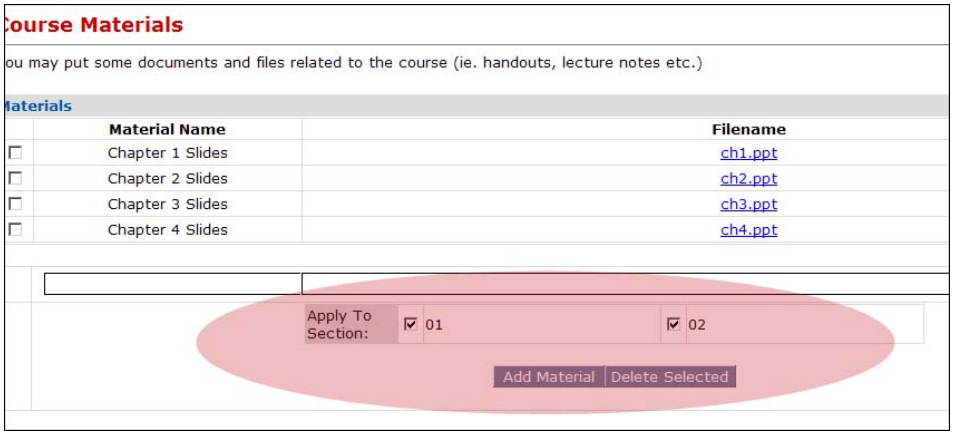

Figure A.15 Coordination of several sections

Another advanced feature of Course ON-LINE is the privacy management. In the "Edit Privacy" page, instructors can define which functions of their course homepage will be available for public use. (Figure A.16)

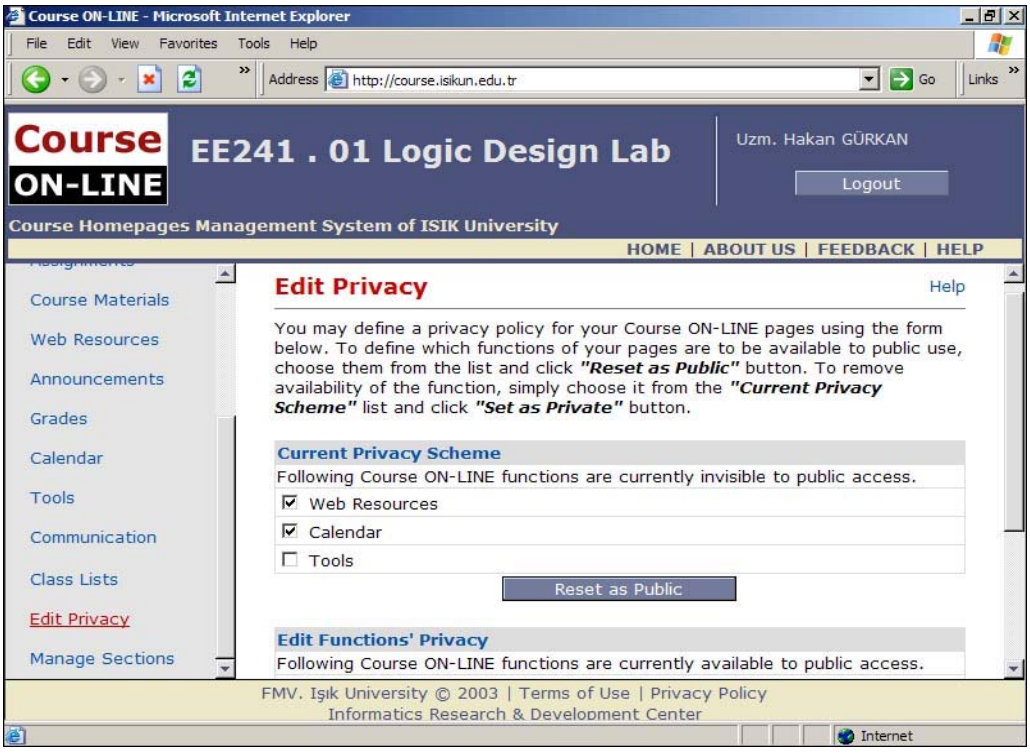

Figure A.16 Edit Privacy

APPENDIX B

AGENT SOURCE CODES

B.1. Source Code of the GAIA Agent

```
import jade.core.*;
import jade.wrapper.*;
import java.sql.*;
import java.util.*;
import jade.core.behaviours.TickerBehaviour;

public class Gaia extends jade.core.Agent {

    /** Creates a new instance of Gaia */
protected void setup() {
      System.out.println("Gaia says : Gaia Started, creating a titan;");
        // Add the TickerBehaviour (period 3 sec)
        addBehaviour(new TickerBehaviour(this, 3000) {
            protected void onTick() {
                System.out.println("Agent "+myAgent.getLocalName()+":
tick="+getTickCount());
                DBConnector con = new DBConnector();
                Vector newAgent = con.getNewAgentInfo();
                int i = 0;
                while (i < newAgent.size()) {
                    Vector tmpvec = (java.util.Vector) newAgent.get(i);
                    String agentId = tmpvec.get(0).toString();
                    System.out.println("New Agent is: "+agentId);

                    // New Titan creation and mark to DB as started
                    try {
                        Object args[] = new Object[1];
                        args[0]=agentId;
                        jade.wrapper.PlatformController cont =
getContainerController();
                        cont.createNewAgent("Titan_"+agentId, "Titan",
args).start();
                        System.out.println("Gaia says : Titan created.");
                        con.setNewAgentAsStarted(agentId);
                    }
                    catch (Exception ex) {
                        ex.printStackTrace();
```

```
                    }
                    i++;
                }
                if (i==0) {
                    System.out.println("No New Agent request arrived");
                }
            }
        });
    }
}
```

B.2. Source Code of the Titan Agent

```
import jade.core.*;
import jade.wrapper.*;
import java.util.*;
import jade.core.behaviours.TickerBehaviour;

public class Titan extends jade.core.Agent {
    Vector agentMonitors;
    String studentName;
    String studentEmail;
    String monitoredCourses;
    String aid;

    /** Creates a new instance of Titan */
    public void setup() {
        aid = "";
        Object[] args = getArguments();
        if(args.length > 0) {
            aid = (String)args[0];
        }

        DBConnector con = new DBConnector();
        studentName = con.getStudentName(aid);
        studentEmail = con.getStudentEmail(aid);

        System.out.println("Titan says : I am agent of: "+studentName);


        addBehaviour(new TickerBehaviour(this, 5000) {
            protected void onTick() {
                String message;
                DBConnector con2 = new DBConnector();
                DBConnector con3 = new DBConnector();
                agentMonitors = con2.getAgentMonitorInfo(aid);
```

```
                monitoredCourses = con2.getMonitoredCourses(aid);
                int i = 0;
                while (i < agentMonitors.size()) {
                    Vector tmpvec = (Vector) agentMonitors.get(i);
                    String functionId = tmpvec.get(0).toString().trim();
                    String lastItem  = tmpvec.get(1).toString().trim();
                    message =
con3.checkCourseOLFunction(functionId,aid,lastItem,monitoredCourses);
                    if (message.length()>0) {
                            System.out.println(message);
                            ms.SendMail sm = new ms.SendMail(studentEmail,
"CourseOL Agent Notification", message);
                    }
                    i++;
                }
                if (con2.isAgentDead(aid)) {
                    kill();
                }
            }

        });
    }
    public void kill() {
        System.out.println("I am dead");
        this.doDelete();
    }
}
```

B.3. Source Code of the GAIA Agent Data Layer

```
import java.sql.*;
import java.util.*;

public class DBConnector {

    public void DBConnector() {
    }
    public Connection getConnection () {
        Connection con;
        try {
            Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
```

```
            con =
DriverManager.getConnection("jdbc:microsoft:sqlserver://irdc:1433;DatabaseNam
e=courseonline", "dbuser", "dbuserpassword");
            return con;
        }
        catch (ClassNotFoundException ce) {
            System.out.println("Class Not Found");
            return null;
        }
        catch (SQLException se) {
            System.out.println(se.getMessage());
            return null;
        }

    }

    public Connection getConnectionTOCOL () {
        Connection con;
        try {
            Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
            con =
DriverManager.getConnection("jdbc:microsoft:sqlserver://poseidon:1433;Databas
eName=CommonDB", "dbuser", "pass");
            return con;
        }
        catch (ClassNotFoundException ce) {
            System.out.println("Class Not Found");
            return null;
        }
        catch (SQLException se) {
            System.out.println(se.getMessage());
            return null;
        }

    }

    public boolean releaseConnection (Connection con) {
        try {
            con.close();
            return true;
        }
            catch (SQLException se) {
                return false;
        }
    }

    public Vector getNewAgentInfo () {
        Connection con;
        DBConnector connector = new DBConnector();
        Vector newAgent = new Vector();
```

```
        try {
            con = connector.getConnection();
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Agent_Log WHERE
isStarted=0");
            while (rs.next()) {
                Vector row = new Vector();
                row.add(rs.getString(1));
                row.add(rs.getString(2));
                newAgent.add(row);
            }
            con.close();
            return newAgent;
        }
        catch (SQLException se) {
            System.out.println(se.getMessage());
            return null;
        }
    }

    public boolean setNewAgentAsStarted(String aid) {
        Connection con;
        DBConnector connector = new DBConnector();
        try {
            con = connector.getConnection();
            Statement stmt = con.createStatement();
            stmt.executeUpdate("UPDATE Agent_Log SET isStarted=1 WHERE
agent_id='"+ aid +"'");
            con.close();
            return true;
        }
        catch (SQLException se) {
            System.out.println(se.getMessage());
            return false;
        }
    }


    public Vector getAgentMonitorInfo(String aid) {
        Connection con;
        DBConnector connector = new DBConnector();
        Vector agentMonitor = new Vector();

        try {
            con = connector.getConnection();
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT fnid,
ISNULL(last_found_id,0) AS lastid FROM Agent_Monitor WHERE agent_id = '"+ aid
+"'");
            while (rs.next()) {
                Vector row = new Vector();
```

```
                row.add(rs.getString("fnid"));
                row.add(rs.getString("lastid"));
                agentMonitor.add(row);
            }
            con.close();
            return agentMonitor;
        }
        catch (SQLException se) {
            System.out.println(se.getMessage());
            return null;
        }
    }


    public String getStudentName(String aid) {
        Connection con;
        DBConnector connector = new DBConnector();
        String stuName="";
        String sql="";

        try {
            con = connector.getConnectionTOCOL();
            Statement stmt = con.createStatement();
            sql = "SELECT * FROM Student WHERE StuID = '"+ aid +"'";
            ResultSet rs = stmt.executeQuery(sql);

            while (rs.next()) {
                stuName  = rs.getString("Stu_FName") + " " +
rs.getString("Stu_LName");
            }
            con.close();
            return stuName;
        }
        catch (SQLException se) {
            System.out.println(se.getMessage());
            return null;
        }
    }

    public String getStudentEmail(String aid) {
        Connection con;
        DBConnector connector = new DBConnector();
        String stuEmail="";
        String sql="";

        try {
            con = connector.getConnectionTOCOL();
            Statement stmt = con.createStatement();
            sql = "SELECT Stu_Univ_E_Mail FROM Student_Address WHERE StuID =
'"+ aid +"'";
            ResultSet rs = stmt.executeQuery(sql);
```

```
            while (rs.next()) {
                stuEmail = rs.getString("Stu_Univ_E_Mail");
            }
            con.close();
            return stuEmail;
        }
        catch (SQLException se) {
            System.out.println(se.getMessage());
            return null;
        }
    }

    public String getMonitoredCourses(String aid) {
        Connection con;
        DBConnector connector = new DBConnector();
        String courseList="";
        String sql="";

        try {
            con = connector.getConnection();
            Statement stmt = con.createStatement();
            sql = "SELECT courseid,crssection FROM Agent_Monitoring_Courses
WHERE agent_id = '"+ aid +"'";
            ResultSet rs = stmt.executeQuery(sql);
            while (rs.next()) {
                courseList += "( courseid = '" + rs.getString("courseid") +
"' AND crssection = '"+ rs.getString("crssection") +"' ) OR ";
            }
            if (courseList.length() > 0) {
                courseList = courseList.substring(0,courseList.length() - 4);
            }
            con.close();
            return courseList;
        }
        catch (SQLException se) {
            System.out.println(se.getMessage());
            return null;
        }
    }

    public String checkCourseOLFunction(String functionId,String aid, String
lastItem,String monCourses) {

        Connection con;
        DBConnector connector = new DBConnector();

        String message="";
        String sql="";
        String id="";
        if (functionId.equals("2")) {
            sql = "SELECT ISNULL(MAX(assignmentid),0) as id FROM Assignments
```

```
WHERE assignmentid > "+ lastItem +" AND (" + monCourses + ")";
            message = "Assignment";
        }
        if (functionId.equals("3")) {
            sql = "SELECT ISNULL(MAX(id),0) as id FROM Course_Materials WHERE
id > "+ lastItem +" AND (" + monCourses + ")";
            message = "Course Material";
        }
        if (functionId.equals("4")) {
            sql = "SELECT ISNULL(MAX(id),0) as id FROM WebResources WHERE id
> "+ lastItem +" AND (" + monCourses + ")";
            message = "Web Resource";
        }
        if (functionId.equals("5")) {
            sql = "SELECT ISNULL(MAX(announcementid),0) as id FROM
Announcements WHERE announcementid > "+ lastItem +" AND (" + monCourses +
")";
            message = "Announcement";
        }
        if (functionId.equals("6")) {
            sql = "SELECT ISNULL(MAX(id),0) as id FROM Grades WHERE stuid =
'"+ aid +"' AND id > "+ lastItem +" AND (" + monCourses + ")";
            message = "Grade";
        }
        if (functionId.equals("7")) {
            sql = "SELECT ISNULL(MAX(eventid),0) as id FROM Calendar WHERE
eventid > "+ lastItem +" AND (" + monCourses + ")";
            message = "Calendar";
        }
        if (functionId.equals("8")) {
            sql = "SELECT ISNULL(MAX(toolid),0) as id FROM Tools WHERE toolid
> "+ lastItem +" AND (" + monCourses + ")";
            message = "Tool";
        }


        try {
            con = connector.getConnection();
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(sql);
            if (rs.next()) {
                id = rs.getString("id");
            }
            con.close();

            if (Integer.parseInt(id) > 0) {
                connector.setFnIdMonitored(id,aid,functionId);
                message = "New " + message + " detected by your Titan Agent.
Please check Course ON-LINE";
            }
            else {
```

```
                message = "";
            }


            return message;
        }
        catch (SQLException se) {
            System.out.println(se.getMessage());
            return null;
        }
    }



    public boolean setFnIdMonitored(String id,String aid,String functionId) {
        Connection con;
        DBConnector connector = new DBConnector();
        try {
            con = connector.getConnection();
            Statement stmt = con.createStatement();
            stmt.executeUpdate("UPDATE Agent_Monitor SET last_found_id='"+ id
+"' WHERE agent_id='"+ aid +"' AND fnid="+ functionId);
            con.close();
            return true;
        }
        catch (SQLException se) {
            System.out.println(se.getMessage());
            return false;
        }
    }

    public boolean isAgentDead(String id) {
        Connection con;
        DBConnector connector = new DBConnector();
        boolean res=true;
        try {
            con = connector.getConnection();
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM Agent_Log WHERE
agent_id='"+ id +"'");
            if (rs.next()) {
                res=false;
            }
            con.close();
            return res;
        }
        catch (SQLException se) {
            System.out.println(se.getMessage());
            return false;
        }
    }
```

```
}
```

B.4. Source Code of E-Mail API Used by GAIA

```
package ms;

import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;

public class SendMail {

  private String to;
    private String message;
    private String subject;


    /** Creates a new instance of SendMail */
    public SendMail(String to, String subject, String message) {
        this.to = to;
        this.message = message;
        this.subject = subject;
        this.sender();
    }

    public void sender () {

        //Create Properties
        Properties props = new Properties();
        props.put("mail.transport.protocol", "smtp");
        props.put("mail.smtp.auth", "true");
        props.put("mail.debug", "smtp");
        props.put("mail.smtp.host", "irdc.isikun.edu.tr");
        props.put("mail.from", "courseonline@irdc.isikun.edu.tr");
        props.put("mail.user", "courseonline@irdc.isikun.edu.tr");

        //Create Authenticator

        //Create JavaMail Session
        Session ses = Session.getDefaultInstance(props, new
SMTPAuthenticator());

        //Make message
        try {
            MimeMessage msg = new MimeMessage(ses);
            InternetAddress from = new
InternetAddress("courseonline@irdc.isikun.edu.tr");
```

```
            from.setPersonal("Course ON-LINE <NO-REPLY>");
            InternetAddress to = new InternetAddress(this.to);

            msg.setFrom(from);
            msg.setRecipient(MimeMessage.RecipientType.TO, to);
            msg.setSubject(this.subject);
            msg.setContent(this.message, "text/plain");

            Transport trans = ses.getTransport();
            trans.connect();
            trans.send(msg);

        }
        catch (MessagingException mx) {
            mx.printStackTrace();
        }
        catch (java.io.UnsupportedEncodingException uue) {
            uue.printStackTrace();
        }
    }
    public String test() {
        return this.to;
    }
    class SMTPAuthenticator extends Authenticator {

        public PasswordAuthentication getPasswordAuthentication() {

            String userid = "courseonline@irdc.isikun.edu.tr";
            String password = "smtppass";

            return new PasswordAuthentication (userid, password);
        }
    }
}
```

C. CD INCLUDING DOCUMENTS AND APPLICATION SOURCE CODES