Dasgupta, P., & Khazanchi, D. (2005). Adaptive decision support for academic course scheduling using intelligent software agents. *International Journal of Technology in Teaching and Learning*, *1*(2), 63-78.

Adaptive Decision Support for Academic Course Scheduling Using Intelligent Software Agents

Prithviraj Dasgupta Deepak Khazanchi

University of Nebraska at Omaha

Academic course scheduling is a complex operation that requires the interaction between different users including instructors and course schedulers to satisfy conflicting constraints in an optimal manner. Traditionally, this problem has been addressed as a constraint satisfaction problem where the constraints are stationary over time. In this paper, we address academic course scheduling as a dynamic decision support problem using an agent-enabled adaptive decision support system. In this paper, we describe the Intelligent Agent Enabled Decision Support (IAEDS) system, which employs software agents to assist humans in making strategic decisions under dynamic and uncertain conditions. The IAEDS system has a layered architecture including different components such as a learning engine that uses historic data to improve decision-making and an intelligent applet base that provides graphical interface templates to users for frequently requested decision-making tasks. We illustrate an application of our IAEDS system where agents are used to make complex scheduling decisions in a dynamically changing environment.

Keywords: Course Scheduling, Adaptive Decision Support System, Software Agents

The advent of the Internet has enabled interaction between users using different formats such as text and multimedia, over geographically dispersed areas, for solving complex problems. Rapid interaction between users over the Internet has already automated various processes including file-sharing and e-commerce. In this paper, we

Prithviraj Dasgupta is an Assistant Professor of Computer Science in the Computer Science Department at the University of Nebraska at Omaha. Deepak Khazanchi is Peter Kiewet Distinguished Professor in the Information Systems & Quantitative Analaysis Department at the University of Nebraska at Omaha. Please contact Dr. Dasgupta College at the College of Information Science & Technology, University of Nebraska at Omaha, Omaha, NE 68182, E-mail: pdasgupta@mail.unomaha.edu

address the problem of academic course scheduling in a networked environment. Academic course scheduling constitutes a complex problem that requires the interaction between different users including instructors and course schedulers to resolve conflicting constraints in an optimal manner. Traditionally, this problem has been addressed as a constraint satisfaction problem where the constraints are available at the central location that performs the course scheduling. Here, we address academic course scheduling in a networked environment using intelligent agents within a decision support framework.

Decision Support Systems (DSS) comprise software systems that assist humans in making complex decisions in real-life problem domains. With the advent of the Internet and powerful computing devices over the last decade, dynamic and intelligent decision support is rapidly emerging as the new research direction in the field of decision support systems. Decision-making problems in real life are characterized by complex, unstructured nature of problem domains, unpredictable outcome of decisions due to the dynamic nature of problems and information, and the potential risks associated with making an incorrect/inaccurate decision. In such a scenario, a naive model that uses a stationary mapping from situation to decision is inadequate for making correct decisions. The information from prior decisions needs to be adapted to the constraints and parameters specified by the current environment to make an accurate decision in a dynamic scenario. With the development of software technologies such as intelligent agents, it is now possible to address the fundamental challenge of combining real-time environmental data with existing decision rules and historical knowledge about the domain obtained from previous experience. In this paper, we address the problem of dynamic decision making using a software agent enabled adaptive DSS that combines real-time environmental data with existing decision rules and historical knowledge about the domain to engender informed decision making.

The rest of the paper is organized as follows. In the next section, we establish the background of this research by discussing related work on DSS and software agents. We elaborate on the notion of agent-enabled DSS in the next section and describe our Intelligent Agent-Enabled Decision Support (IAEDS) architecture after that. Subsequently, we illustrate the proposed IAEDS architecture with a detailed discussion of a dynamic academic course scheduling system. In the final section, we summarize this paper and discuss future research directions.

RELATED WORK

DSS are software applications that have been used over the last few decades to provide support for many structured and unstructured problems such as strategic planning, investment planning, stock portfolio management, enterprise planning, human resources management, supply chain planning, knowledge management, case-based reasoning and help desk automation (Clemen, 1996; Mallach, 2000; Marakas, 1998; Mora, Forgionne, & Gupta, 2002; Turban and Aronson, 1997). DSS components such as knowledge management systems, model management systems and data management systems aid humans in making better decisions by incorporating previous knowledge and information about the domain. Over the past two decades, decision support systems and software agent enabled systems have been researched independently both in academia and in commercial applications. Several businesses have successfully implemented decision support systems to solve problems including human resources management, supply chain planning, help desk automation, and placement of new office locations. Most of these decision support systems use static models of the problem domain because the underlying data in these applications does not change drastically over time. Some dynamic decision models are used in applications such as sales forecasting, predicting consumer responses and deciding company strategies. Decision support using dynamic models is more complicated because it involves parameters that change over time and are difficult to predict or estimate beforehand. Although dynamic decision support systems have been developed successfully, not many applications exist that intelligently incorporate the dynamics of a real-time setting into the decision making process. Intelligent software agents provide a technology that can be used to obtain knowledge from dynamically changing environments and thus potentially allowing DSS users to make more informed and accurate decisions (Mora et al., 2002).

Software agents (Russell & Norvig, 2003; Weiss, 1999) are used widely in various applications such as searching information on the Web (Dasgupta, Narasimhan, Moser, & Melliar-Smith, 1999; Knapik & Johnson, 1997), tracking browsing behavior of online users (Sahai, Billiart & Morin, 1997), implementing trading algorithms for online auctions (Sandholm, 2002), assisting humans in online activities such as filling forms or presenting information in a concise form (Padgham & Winikoff , 2004), and, even for security and privacy applications such as detecting spyware, implementing security policies on Web servers, and, filtering spam (Dasgupta, Moser & Melliar-Smith, 2000). For example, MySimon.com uses software agents to compare the prices of items from different online sellers while online merchants such as Amazon.com and E-bay employ software agents for adjusting the prices of items dynamically depending on factors such as consumer preferences and market demand. In the Virtual Information Processing Agent Research (VIPAR) project that is targeted for military applications, intelligent software agents are used to extract relevant information from data collected from various sources into a form that can be analyzed by humans (Potok, Elmore, Reed, & Sheldon, 2003). Recently, the LogNet system being developed for Boeing uses decision support techniques enabled by an agent based learning engine to determine re-supply decisions for fuel, ammunition and medical supplies during wartime. The real-time information obtained from the battlefield is used to improve logistics by observing correlations between current situations and the outcomes of past re-supply decisions.

Academic course scheduling or timetabling has been traditionally viewed as a constraint satisfaction problem (Blanco & Khatib, 1998; Dignum, Nuijten, & Janssen, 1995). Various techniques including linear programming (Carter, 1986), logic programming (Frangouli, Harmandas, & Stamatopoulos, 1995; Stamatopoulos, Viglas & Karaboyas, 1998), genetic algorithms (Burke, Elliman & Weare, 1995; Elmohamed, Coddington, & Fox, 1998), self adaptive algorithms (Socha, Sampels, & Manfrin, 2003) and heuristic-based approaches (Burke, Elliman, & Weare, 1994; Lewandowski & Condon, 1996) have been used to resolve conflicts in course scheduling problems. However, most of these algorithms assume that the constraints are already available at the central location performing the course scheduling. In this paper, our focus is on addressing course scheduling problems in a distributed environment within the framework of a decision support system using software agents.

DYNAMIC DECISION SUPPORT USING SOFTWARE AGENTS

A dynamic environment is characterized by non-deterministic and possibly rapid changes. DSS operating in dynamic environments should therefore adapt the decision making procedure to the current parameters and constraints of the real-time environment to assist the decision maker in reaching an accurate and effective decision. Making the correct decision can be looked upon as solving a constraint satisfaction problem given the relevant historical information and a set of parameters describing the current environment. For complex applications, the solution of this problem can become quite involved. Therefore, it is difficult, and at times even impossible, for humans to make correct decisions without any computational aid. Software agents provide a suitable paradigm for automating complex tasks and solving complex problems more accurately and rapidly than humans. Software agents can enhance traditional DSS by rapidly updating and using knowledge and domain information from a DSS so that the agents can respond efficiently and accurately to user queries.

An agent is a software entity that can autonomously perform the tasks that have been encoded into it without continuous supervision (Bigus & Bigus, 2001; Bradshaw, 1997; Weiss, 1999, Wooldridge, 2002). Besides being autonomous, a software agent is characterized by the following key features:

- *Goal-directed*: An agent can be provided with a goal such as making a decision on a particular attribute of a system. The tasks that are encoded into the agent enable it to work towards that goal.
- *Reactive*: An agent responds to its queries or requests from its environment and takes responds to those queries through actions.
- *Pro-active*: An agent is also self-motivated to perform actions that help it to achieve its goal more efficiently.

Software agents can also be adapted to provide support for strategic decision-making and/or semi-structured problem solving. For example, a software agent can be programmed to dynamically learn system parameters and use these parameters to improve or evolve its actions so that it can reach its goal more efficiently. However, the knowledge that an intelligent agent acquires during execution cannot be stored beyond its lifetime. Discarding this knowledge would also be inappropriate as later decisions might require the experience gained by previous agents. An intelligent agent is frequently augmented with a knowledge base to store the experiences it acquires from the environment (Bui & Lee, 1999; Power, 2000). DSS contain components such as knowledge management subsystems, data management subsystems and model management subsystems that store domain data, knowledge and rules for enabling the decision process (Turban, 2001). In consequence, an intelligent agent enabled decision support system can potentially store the knowledge gained by software agents in a knowledge base along with existing rules governing the domain and data about the environment.

IAEDS SYSTEM ARCHITECTURE

The objective of our proposed Intelligent Agent Enabled Decision Support (IAEDS) system architecture is to combine the advantages from the domains of software agent enabled computing and decision support systems. Traditional DSS comprise knowledge and rules that are static in nature. The knowledge gained by software agents can be used to dynamically update the information stored in a DSS so that it can respond to user queries and requests more efficiently. Our proposed IAEDS system architecture combines software agent based computation with a dynamically updated DSS. The software model for the IAEDS system is illustrated in Figure 1. The functionality of the system can be broadly categorized into two separate layers; viz., a *software agent layer* that encapsulates the operation of agents and a *decision support layer* that contains the modules for decision support.

SOFTWARE AGENT LAYER

The software agent layer contains the tasks performed by the intelligent software agents in the IAEDS system. These tasks include:

- Extract and filter the user query.
- Translate the user query from a Web-based markup language such as XML into a domain specific query language understandable by agents. We propose to use FIPA ACL (FIPAACL, 2004) as the language for interactions between our agents. FIPA ACL is a performative based language that has been proposed and adopted by the Foundation for Intelligent Physical Agents (FIPA) as the standard for interoperations between heterogeneous software agents.
- Create and execute agents to perform the tasks requested by the user.
- Store reusable agents in an agent repository for performing frequently requested tasks.
- Assimilate the knowledge gathered by the agents to improve the system's intelligence and decision-making capabilities.

Figure 1. High-level Schematic for IAEDS System

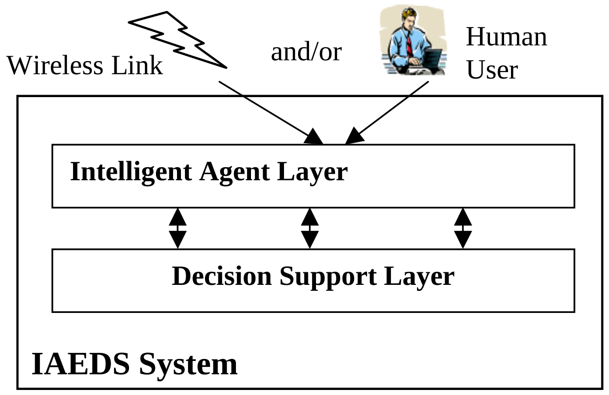
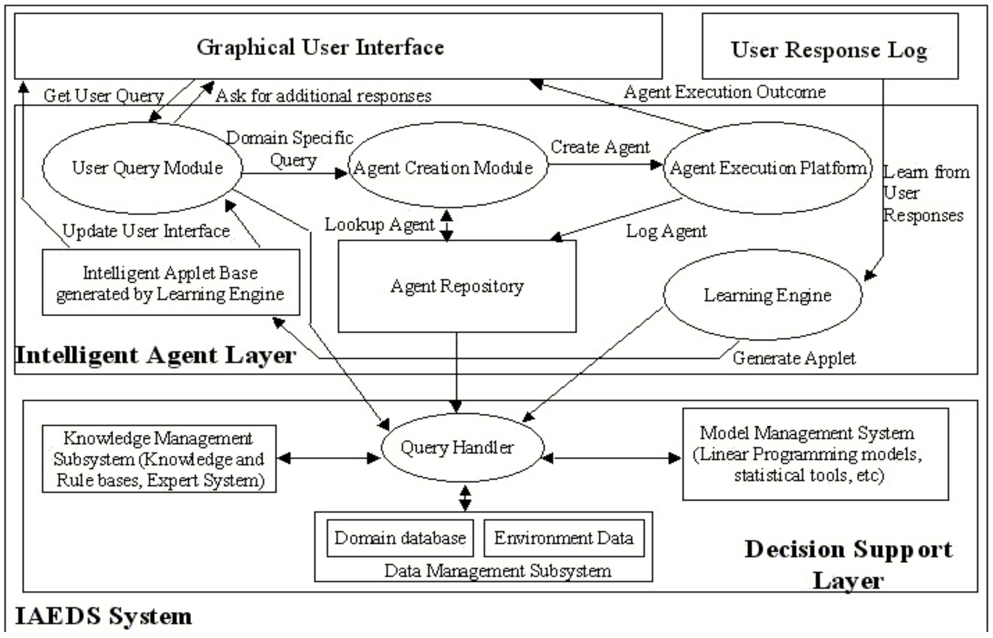


Figure 2. Detailed IAEDS System Architecture



The different modules designed to perform these tasks are illustrated in the IAEDS system architecture shown in Figure 2. The software agent layer is initially equipped with software agents that are capable of performing common tasks in the problem domain. The agents learn from their actions (Mitchell, 1997) and update the agent repository, and, data and knowledge bases in the decision support layer while the system operates. As the knowledge about the domain increases, the agents are dynamically evolved to perform tasks that are more complex. The specific functionality of each of these modules is summarized in Table 1.

Table 1. Functionality of Modules in the Software Agent Layer

Module	Functionality
User Query Module	User enters query through an XML annotated GUI. A query parsing agent converts the query into the FIPA ACL syntax The query-parsing agent then validates the query for consistency by accessing the domain and knowledge management subsystems. The validated query is forwarded to the agent creation module.
Agent Creation Module	Creates an agent for performing the specific action requested in the query. Looks up in the agent repository to detect previously created agents with similar functionality as that desired in the current query. If such an agent is found, it is cloned from the agent repository and adapted to execute the current query. Cloning the agent supports asynchronous operation of the system. If an agent with a similar functionality is once again required before the current query completes, the agent from the repository can be cloned again without waiting for the agent executing the current query to complete its operation. If an agent with the desired functionality is not found in the repository, a new agent is created and its information is entered in the agent repository.
Agent Execution Platform	An agent runs on the agent execution platform to perform tasks outlined in user-query. The agent needs to access the components in the decision support layer to determine whether its actions are consistent with the domain knowledge and rules. After execution, the agent is stored in the agent repository. The result of agent execution is returned to user through the GUI.
User Response Log	The user expresses his or her level of satisfaction with the query from the results returned by the agent on a predetermined scale. User responses are logged in the user response log along with the identity of the agent that executed the query. The contents of the user response log are used to update the knowledge management subsystem in the decision support layer.
Agent Repository	Contains agents created by the agent creation module. Description of its behavior, Actions taken by the agent during execution and Responses returned by users after the agent's execution.
Learning Engine	Uses machine-learning techniques to combine information about the agents stored in the agent repository and information about the domain from the decision support layer. Updates and refines the facts and rules stored in the knowledge management subsystem within the decision support layer Uses domain knowledge that it gathers from the software agents and the knowledge management subsystem to create intelligent applets that aid the user. Applets are equipped with the knowledge obtained from executing tasks requested by previous users of the IAEDS system; they present a more intelligent and educated interface to the user than the basic GUI.
Intelligent Applet Repository	Contains applets generated by the learning engine. As new applets are generated, they are made accessible to the user Agents in the user query-parsing module also check the applet repository while performing consistency checks on user queries.

DECISION SUPPORT LAYER

The decision support layer performs the traditional functions of a DSS and also adapts and improves itself from the results returned by the agents in the software agent layer. As shown in Figure 2, this layer comprises a data management subsystem, a model management subsystem, and a knowledge management subsystem. The functionality of each of these subsystems is summarized in Table 2.

Table 2.Functionality of Modules in the Decision Support Layer

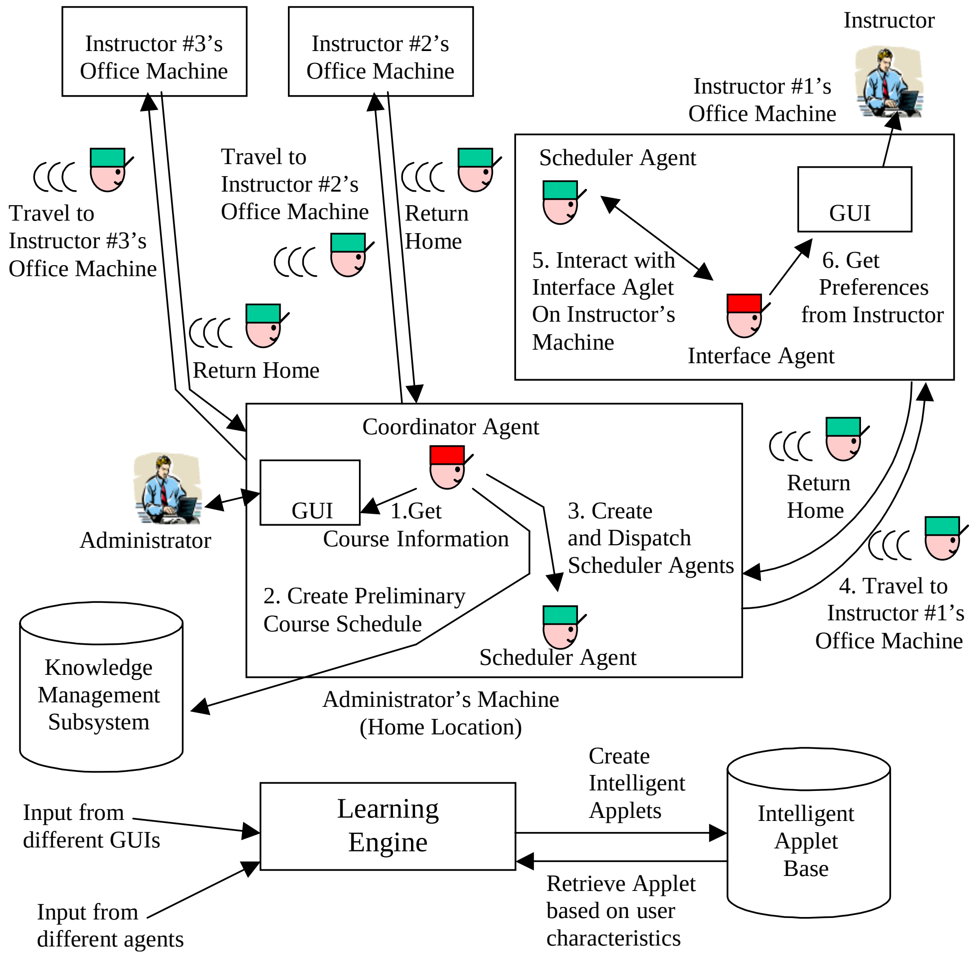
Module	Functionality
Data Management Subsystem	Comprises a database for storing the data and information gathered from the environment by the software agents, and a system to access that data through a query tool.
Model Management Subsystem	Encapsulates the analytical model employed by the decision support layer to solve the decision-making problem. Different parameters associated with the decision-making problem are provided as input and the model finds the optimal solution to the problem for the present set of inputs. The model is refined dynamically with the feedback given by the users so that it can predict the optimum result more accurately.
Knowledge Management Subsystem	Contains the knowledge and rules for the domain. Initially, the knowledge and rule bases are provided with information obtained from historical data or from simulation runs. As software agents gain information from the environment, the knowledge base is updated with new knowledge and rules.
Query Handler	Accepts queries sent to the decision support layer from the software agent layer and hands it over to the appropriate subsystem within the decision support layer. Obtains the relevant information from all the subsystems in the decision support layer and returns a single coherent reply to the software agent layer.

AN ACADEMIC COURSE SCHEDULING SYSTEM USING IAEDS

In academic environments such as school, college, and university courses that are offered during a particular term have to be assigned to instructors. The person coordinating the course assignments also has to ensure that there are rooms available to teach the courses, match the contents of courses with the teaching interests of different instructors and resolve possible conflicts between different course schedules. The *course assignment problem* can therefore be viewed as a non-trivial constraint satisfaction problem. In our IAEDS-enabled course scheduling system, we use software agents that utilize information including the course schedules of previous terms, the estimated number of students for every course, availability of different classrooms, and the research and teaching interests of different instructors, to develop a preliminary assignment of courses to instructors. A mobile scheduling agent then requests every instructor to respond to the preliminary course assignments. The final course schedule is determined by resolving any conflicts that might have arisen after assimilating the instructors' responses.

We use IBM Aglets (Lange, D. & Oshima, 1998) to implement our course scheduling system. Aglets are Java enabled mobile agents that encapsulate the program logic and can be transported using HTTP between different computers. A computer should run an aglet server on a specified port to be able to send, receive and host aglets. An aglet is implemented as a Java thread that can be stopped, packaged into bytecode and transferred to a remote site running an aglet server. On reaching the remote site, the aglet’s bytecode is deserialized and the aglet's execution thread is resumed. A mobile aglet visiting a remote site can interact with the remote site through intermediaries in the form of stationary aglets to extract information and perform tasks remotely.

Figure 3: *Schematic of operation for an IAEDS course scheduling system*


Instructor #3's Office Machine
Instructor #2's Office Machine
Instructor
Instructor #1's Office Machine
Scheduler Agent
GUI
Travel to Instructor #3's Office Machine
Travel to Instructor #2's Office Machine
Return Home
5. Interact with Interface Aglet On Instructor's Machine
6. Get Preferences from Instructor
Interface Agent
Return Home
Coordinator Agent
GUI
1.Get Course Information
3. Create and Dispatch Scheduler Agents
Return Home
Administrator
2. Create Preliminary Course Schedule
4. Travel to Instructor #1's Office Machine
Scheduler Agent
Knowledge Management Subsystem
Administrator's Machine (Home Location)
Create Intelligent Applets
Input from different GUIs
Learning Engine
Intelligent Applet Base
Retrieve Applet based on user characteristics
Input from different agents

EXTRACTING INSTRUCTORS' PREFERENCES

As shown in Figure 3, human users interacting through aglet servers running on remote machines represent the instructors in our system. The steps for extracting the instructors' preferences are the following:

1. A central administrator site is used to initiate and coordinate the activities in the course scheduling system. The administrator site contains a stationary *coordinatorAgent* that obtains the information about the courses to be offered for the current term from the human course administrator through a GUI.
2. The *coordinatorAgent* then accesses the database within the Knowledge Management Subsystem (KMS). The database contains information about previous course schedules including instructor assignments, teaching preferences, preferred class timings, and student enrollment. For existing courses, the *coordinatorAgent* utilizes the course information already present in the KMS. For new courses, or courses that require an instructor or schedule change from

previous offerings, the *coordinatorAgent* determines the instructor that is best qualified to teach the course from the KMS.

3. After gathering the relevant information from the KMS, the *coordinatorAgent* creates one mobile *schedulerAgent* for every instructor available during the current term. The *schedulerAgent* carries within it the preliminary course allotment for the instructor and information about other courses that the instructor is eligible to teach for the current term.
4. When the *schedulerAgent* arrives on an instructor's machine, it interacts with a stationary *interfaceAgent* on the machine and displays a preliminary course allotment through a GUI. The instructor can respond in one of the following ways:
    - Accept all the course allotments without making any changes.
    - Respond indicating the attributes such as time or classroom for one or more courses that are unacceptable.
    - Respond with a prioritized list of courses including courses that are not allotted to the instructor, but the instructor is eligible to teach.
5. After the *schedulerAgent* obtains a response from the *interfaceAgent* of the instructor site, it reverts to the administrator site.
6. After all the *schedulerAgents* that had been dispatched to the different instructors' machines return to the administrator’s site, the *coordinatorAgent* extracts the instructors' responses. At the end of this step, the instructors’ preferences from the different instructors are available on the administrator site. The *coordinatorAgent* then proceeds to resolve potential conflicts between the different instructors’ preferences.

RESOLVING CONFLICTS BETWEEN INSTRUCTORS’ PREFERENCES

Conflicts can arise between instructors’ preferences when two or more instructors request simultaneous and exclusive access to the same resource such as courses, classrooms, class timings, etc. For example, two or more instructors might prefer to teach at the same time in the same room, multiple instructors might prefer to teach the same course, or, an instructor might prefer to have more students in a course than the capacity of the room allotted for the course. In all the above cases, the conflicts need to be resolved so that the resources are requested uniquely by each instructor.

We have identified the following two types of potential conflicts in our system:

1. Single Conflict: This type of conflict relates to a single preference.
2. Multi Conflict: This type of conflict exists across two or more preferences.

Multi-conflicts are more difficult to resolve than single conflicts. We should also resolve single conflicts carefully by ensuring that the resolution does not create a new multi-conflict.

Conflict resolution between instructors’ preferences is done in two steps in our system by a *conflictResolutionAgent*. The first step utilizes the domain rules that are contained within the KMS along with an inferencing technique to identify and resolve conflicts. Most single conflicts and some multi-conflicts can be resolved during this step. For conflicts that still remain unresolved at the end of the first step, the *conflictResolutionAgent* probabilistically selects one instructor preference out of a conflicting set of preferences using a score based mechanism for preferences.

COURSE SCHEDULING DOMAIN RULES

The rules in the KMS enforce operational constraints that ensure that different courses in the current schedule satisfy certain criteria including timing requirements, enrollment size restrictions, and also ensure that preferences from different instructors do not conflict in content or schedule with one another. The entire set of rules and conflict resolution among the preference is beyond the scope of this paper. Therefore, for describing the operation of our system we have selected some basic rules for course scheduling as shown below:

- Rule 1: If two instructors prefer the same classroom at the same time then assign a different room for one of the instructors. If an alternate room is not available at the same time then change the timing of one of the courses.
- Rule 2: If a classroom cannot hold the number of students an instructor prefers in a course, then limit the enrollment size to the classroom's capacity.
- Rule 3: Classes for graduate level courses must be two hours in duration and must be taught in the evening, starting between 16:00 and 18:00 hours.

RESOLVING THE CONFLICTS USING THE DOMAIN RULES

To illustrate the operation of our inferencing technique using the rules shown above we have assumed that there are five instructors' preferences, which have to be inspected for potential conflicts. For simplicity, we have shown only the relevant attributes for every preference in Table 3. In addition, it is assumed that the preferences relate to courses taught on the same day.

Table 3. Instructors' preferences from five different instructors.

Preference ID	*Instructor ID*	*Course ID*	*Room Number*	*Timing*	*Enrollment Size*	*Room Capacity*	*Course Level*
1	I002	15368	352	16:00	25	10	Grad
2	I003	25678	352	18:00	45	20	Grad
3	I005	16872	158	08:00	38	30	Grad
4	I006	23417	352	16:00	20	40	Grad
5	I009	36128	278	16:00	30	50	Grad

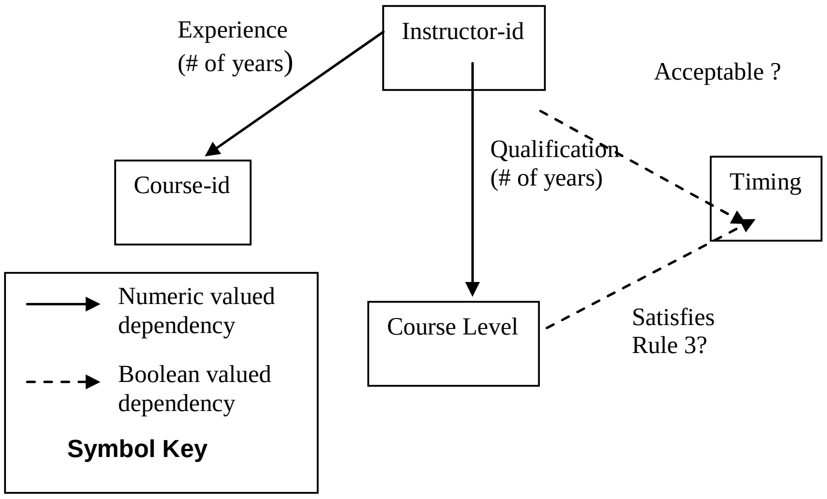
The conflicts for the instructors' preferences shown in Table 3 are as follows:

- *Conflict #1:* For preferences 1, 2, 3 and 5 the preferred enrollment size is greater than the room capacity of the allotted room. This violates rule 2. Each of these preferences contains a single conflict because the unacceptable value of the attribute enrollment size occurs only within its related preference.
- *Conflict #2:* The timing of the graduate class in preference 5 is at 8 am. This is also a single conflict because it violates rule 3.
- *Conflict #3:* Preferences 1 and 4 occupy the same room at the same time and violates rule 1. This is a multi conflict because each preference when considered independently is acceptable but when the two preferences are considered simultaneously, they represent a conflict.

The single conflicts are first chosen by the *conflictResolutionAgent* and resolved. For conflict #1, the preferred enrollment size in each preference is reduced to the room capacity according to rule 1. This is a trivial resolution and no new multi-conflicts are created due to the resolution. Similarly, conflict #2 is resolved by changing the time of the course to an evening time. This resolution also does not create a new multi-conflict with the existing preferences.

Next, the *conflictResolutionAgent* tries to resolve multi-conflict #3. The two courses represented by preferences 1 and 4 cannot be offered simultaneously because they are preferred to be taught at the same classroom at the same time. A trivial solution to the problem can be to assign a later time such as 18:00 hours for one of the courses. However, this creates another multi-conflict with preference 2.

Figure 4. Dependencies between selective attributes of a preference



To illustrate the operation of our system for conflicts that cannot be resolved using rule-based inferencing, let us assume that there are no other classrooms available at the allowed times for preferences 1 and 4. Therefore, only one of the courses in preferences 1 and 4 can be offered. Conflicts of this type cannot be resolved using inferencing, instead the *conflictResolutionAgent* determines a score for every preference. The relevant attributes for the preference are represented as nodes in a graph while edges in the graph represent the dependency between a pair of attributes. Each edge is associated with a weight and a value for the dependency represented by the edge. The value of a dependency is determined from the KMS. For example and as shown in Figure 4, information about the instructor such as experience in teaching a particular course, suitability of class timings and qualification to teach a particular course level are contained within the table containing instructor information within the KMS. Similarly, the suitability of the timing information for a course level is contained within the domain rule base of the KMS. The human course administrator initially determines the weight associated with an edge. These weights are adaptively learnt by the *conflictResolutionAgent* and adjusted according to the course requirements. The score for a preference is determined by the *conflictResolutionAgent* using the pseudo-code algorithm shown in Figure 5. The preference that gets the highest score is selected as the course to be offered in the final course schedule.

After all the conflicts between the instructors' preferences have been removed, the preferences are ready to be allocated as the final course schedule. The *coordinatorAgent* then creates a mobile *courseAllotmentAgent* that visits every instructor's machine and displays the courses and the related attributes assigned to the respective instructors.

Figure 5. Pseudo-code algorithm for calculating the score associated with a preference represented as a graph.

```
double calculatePreferenceScore (Graph g){

   score = 0.0;

   cond = conjunction of boolean valued dependencies;

   if (cond == true)
for every edge with numeric valued dependency
          score = score + (weight of edge) * (value of dependency
```

LEARNING ENGINE OF THE COURSE SCHEDULING SYSTEM

The learning engine in our system is implemented by a *learningAgent* that accepts and analyzes the responses from the *schedulerAgents* and updates different parameters related to the performance of the system. The *schedulerAgents* that visit different instructor machines collect different parameters including the response time required by instructors, the variance between the preliminary course assignments and the preferences expressed by each instructor, and the flexibility of instructors for teaching different courses. The *coordinatorAgent* on the administrator's site extracts these parameters and provides them to the *learningAgent*. The *learningAgent* then attempts to improve and adapt these parameters for use by future *schedulerAgents* created in the system. For example, from the response times of different instructors the *learningAgent* can identify those instructors that are slow in their response and for future course scheduling cycles, it can send periodic reminders to them. The *learningAgent* can also use the variance in the preliminary and final assignments to make preliminary course assignments more intelligently in the future. Finally, by learning the degree of flexibility of different instructors, the *learningAgent* can attempt to assign courses with variable content and schedule to flexible instructors while limiting to standardized courses when responding to the other instructors. The *learningAgent* therefore can improve the intelligence of different agents used in the system by estimating various parameters involved in the course scheduling process and, therefore, potentially increase the efficiency of the system.

USER RESPONSE LOG AND INTELLIGENT APPLET BASE IN THE COURSE SCHEDULING SYSTEM

The interactions between the *interfaceAgent* and the human instructors on different machines are stored in the user response log. The contents of the user response log are reported to the *learningAgent* at certain intervals. The *learningAgent* can use these responses to estimate the behavior of humans using the system. Behavior characteristics can include frequently asked questions, GUI components that are used mostly and commonly encountered problems. The *learningAgent* can then improve the interaction with human users by creating slightly different versions of interfaces for different types of users and storing them as Java applets in an intelligent applet base. The *interfaceAgent* in the system can then select the appropriate applet from the intelligent applet base depending on the user characteristics. For example, an instructor who is comfortable with graphics is displayed an applet with a GUI that contains more graphics and less text as compared to the default GUI. As another example, an applet that contains few navigational instructions can be used to interact with veteran users of the system while applets that contain various usage and navigational hints can be used for novices. The intelligent applets improve the efficiency of the system by reducing the time required by human users to learn and use the software. The variety of applets stored in the intelligent applet base also improves the usability and satisfaction of the different types of human users in the system.

Table 4. Mapping between the Course Scheduling and IAEDS Architecture.

	Common Components	*Course Scheduling System*
Intelligent Agent Layer	Graphical User Interface	GUI for Administrator GUI for Instructor's machines
	User Query module	Logic behind the GUI
	Agent Creation Module	Creates the following agents: Interface Agent Scheduler Agent Coordinator Agent
	Agent Execution Platform	Create and Dispatch Scheduler agents
	User response log	Stores Instructor's responses.
	Agent Repository	Stores the following agents for future reference: Interface Agent Scheduler Agent Coordinator Agent
Decision Support Layer	Learning Engine	Learning Engine Creates Intelligent Applets based on user characteristics
	Query Handler	Query Handler: Brings necessary information from the Knowledge Management subsystem for the previous course schedules, existing agents in database etc.
	Knowledge Management Subsystem	Knowledge Management Subsystem
	Data Management Subsystem	Data Management Subsystem
	Model management Subsystem	Model management Subsystem

CONCLUDING REMARKS

With advancements in the fields of software agents and decision support systems, we envisage that the development of systems combining these two fields will rapidly emerge as an essential technology in the future. Augmenting decision support systems with software agents gives the human user and/or machines the ability to complement historical information with the knowledge acquired from the environment. In this paper, we proposed a generic IAEDS system and demonstrated its architecture within the problem domain of academic course scheduling. Table 4 illustrates the correspondence between the generic proposed IAEDS architecture and the components of the course scheduling system. This prototype academic instructor/course scheduling system demonstrates various facets of the IAEDS architecture including the use of software agents to combine real-time environmental data and decision rules with historical knowledge to provide an adaptive and intelligent DSS.

In the future, we intend to enhance and improve the functionality of our prototype system. For example, future extensions to the course scheduling system include a multi-phase protocol for instructor consensus before finalizing the course schedule. In addition, the research described in this paper also contributes to the development of adaptive learning techniques and evolutionary algorithms, intelligent user-interfaces, integration of software agents in DSS, and implementation of such systems. Finally, the IAEDS architecture is potentially applicable to other more complex problem domains such as transport management, financial portfolio management, commodity trading, and strategic decision support during emergency and combat situations.

REFERENCES

Bigus, J., & Bigus, J. (2001). *Constructing software agents using Java*. New York: John Wiley & Sons.

Blanco, J., & Khatib, L. (1998). "Course scheduling as a constraint satisfaction problem," *Proceedings of PACT98-- The Fourth International Conference and Exhibition on The Practical Application of Constraint Technology*, Wednesday 25th March – Friday 27th March, London, UK.

Bradshaw, J. M., Ed. (1997). *Software agents*. Cambridge, MA: MIT Press.

Bui, T., & Lee, J. (1999). An agent-based framework for building Decision Support Systems. *Decision Support Systems*, *25*(3), 225-237.

Burke, E. K., Elliman, D. G. & Weare, R. F. (1994). A university timetabling system based ON graph colouring and constraint manipulation. *Journal of Research on Technology in Education*, *27*(1), 1-18.

Burke, E. K., Elliman, D. G., & Weare, R. F. (1995). Specialised recombinative operators for the timetabling problem. In T. C. Fogarty (Ed.), *Evolutionary computing, AISB workshop*, UK, (Lecture Notes in Computer Science Series), Springer-Verlag GmbH, 75-85.

Carter, M. W. (1986). A survey of practical applications of examination timetabling. *Operations Research*, *34*(2), 193-202.

Clemen, R. (1996). *Making hard decisions: An introduction to decision analysis.* Belmont, CA: Duxbury Press.

Dasgupta, P, Narasimhan, N, Moser, L., & Melliar-Smith, P. M. (1999). MAgNET: mobile agent based network electronic trading. *IEEE Transactions in Knowledge and Data Engineering*, *24*(6), 509-525.

Dasgupta, P., Moser, L., & Melliar-Smith, M. (2000). The security architecture for MAgNET: A mobile agent e-commerce system. *Proceedings of the Third International Conference on Telecommunications and E-commerce*, Dallas, TX, 289-298.

Dignum, F., Nuijten, W., & Janssen L. (1995). *Solving a time tabling problem by constraint satisfaction*, Technical report, Eindhoven University of Technology.

Elmohamed, M., Coddington, P., & Fox G. (1998). A Comparison of Annealing Techniques for Academic Course Scheduling. In *Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling* (Practice and Theory of Automated Timetabling), Syracuse, NY, 146-166.

FIPAACL (2004). FIPA ACL message structure specification. Retrieved December 18th, 2004, from: http://www.fipa.org/specs/fipa00061.

Frangouli, H., Harmandas, V., & Stamatopoulos, P. (1995). UTSE: Construction of Optimum Timetables for University Courses - A CLP based Approach. In *Proceedings of the 3rd International Conference on the Practical Applications of Prolog PAP '95, Paris*, 225—243.

Knapik, M., & Johnson, J. (1997). *Developing intelligent agents for distributed systems: Exploring architectures, techniques, and applications*. Emeryville, CA: McGraw-Hill Osborne Media.

Lange, D., & Oshima, M. (1998). *Programming and deploying java mobile intelligent with IBM aglets*. Boston, MA: Addison-Wesley Publishing Inc.

Lewandowski G., & Condon A. (1996). Experiments with parallel graph coloring heuristics and applications of graph coloring heuristics and applications of graph coloring, Cliques, Coloring, and Satisfiability. In David S. Johnson and Michael A. Trick (Eds.), *Second DIMACS Implementation Challenge* (*DIMACS Series in Discrete Mathematics and Theoretical Computer Science)*, Providence, RI: American Mathematical Society, *26*, 309—334.

Mallach, E., G. (2000). *Decision support and data warehouse systems*. New York: McGraw-Hil1.

Marakas, G. (1998). *Decision support systems in the 21st century*. Upper Saddle River, NJ: Prentice Hall.

Mitchell, T. (1997). *Machine learning*. New York: McGraw Hill.

Mora, M., Forgionne, G., & Gupta, J. (2002). *Decision making support systems: achievements, trends and challenges for the new decade*. Hershey, PA: Idea Group Publishing.

Padgham, L., & Winikoff, M. (2004). *Developing intelligent agent systems: A practical guide*. Chichester, England: John Wiley & Sons.

Potok, T., Elmore, M., Reed, J., & Sheldon, F. (2003). VIPAR: Advanced information agents discovering knowledge in an open and changing environment. In *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics Special Session on Agent-Based Computing*, Orlando, FL, 28-33.

Power, D. J. (2000). "Web-based and model driven decision support systems: Concepts and issues," In *Proceedings of the Americas Conference on Information Systems*, Long Beach, CA.

Russell, S., & Norvig, P. (2003). *Artificial intelligence: A modern approach*. Upper Saddle River, NJ: Prentice-Hall.

Sahai, A., Billiart, S., & Morin, C. (1997). A portable and mobile manager for distributed system management. In *Proceedings of the Third Joint Conference on Information Sciences*, Raleigh, NC.

Sandholm, T. (2002). Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, *135*, 1-54.

Socha, K., Sampels M., & Manfrin M. (2003). Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In *Proceedings of 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization*, Lecture Notes in Computer Science Series, Springer-Verlag, 2611, 334-345.

Stamatopoulos, P., Viglas, E., & Karaboyas, S. (1998). Nearly optimum timetable construction through CLP and intelligent search. *International Journal on Artificial Intelligence Tools*, *7*(4), 415-442.

Turban, E., & Aronson, J. E. (2001). *Decision support systems and intelligent systems.* Upper Saddle River, NJ: Prentice Hall.

Weiss, G., (1999). *Multiagent systems*. Cambridge, MA: MIT Press.

Wooldridge, M. (2002). *An introduction to multiagent systems*. Chichester, England: John Wiley & Sons.