A MULTIAGENT SYSTEM FRAMEWORK FOR SOLVING THE STUDENT SECTIONING PROBLEM

JOSEPH ANTHONY CARABALLE HERMOCILLA

SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF THE PHILIPPINES LOS BANOS
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE
DEGREE OF

MASTER OF SCIENCE
(Computer Science)

December 2008

The thesis attached hereto, entitled "A MULTIAGENT SYSTEM FRAMEWORK FOR SOLVING THE STUDENT SECTIONING PROBLEM" prepared and submitted by JOSEPH ANTHONY CARABALLE HERMOCILLA in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE (COMPUTER SCIENCE) is hereby accepted.

CONCEPCION L. KHAN
Member, Guidance Committee

_____
Date Signed

JAIME M. SAMANIEGO
Member, Guidance Committee

_____
Date Signed

ELIEZER A. ALBACEA
Chair, Guidance Committee

_____
Date Signed

Accepted as partial fulfillment of the requirements for the degree of MASTER OF SCIENCE (COMPUTER SCIENCE).

ELIEZER A. ALBACEA
Director, Institute of Computer Science

_____
Date Signed

OSCAR B. ZAMORA
Dean, Graduate School

_____
Date Signed

BIOGRAPHICAL SKETCH

The author was born on July 10, 1980 at barangay Malansad Nuevo, Libmanan Camarines Sur. He is the eldest son of Guido V. Hermocilla and Jovina V. Caraballe. During his childhood years, his friends and classmates called him Jo-An. In high school, he is known as Herms. Today, he known to his students as Sir JACH. He still prefers to be called by his first first name, Joseph.

Joseph started elementary school at the Malansad Nuevo Elementary School where he studied until Grade 3. He then transferred to Naga Parochial School in Naga City where he finished his elementary education until Grade 6. He spent his high school years at the Jesuit-managed school, Ateneo de Naga University. After passing the UPCAT, he entered the University of the Philippines Los Banos to start working on his college degree. He was admitted to the BS Forest Products Engineering curriculum in 1996 but transferred to BS Computer Science after two years. Overall, he spent six years in college before obtaining his bachelor's degree in Computer Science in 2002.

At present, he is teaching at the Institute of Computer Science, College of Arts and Sciences, in UP Los Banos. Aside from teaching, he is also doing research in various fields of computer science. His mission in life is to serve others and make them happy using the talents and gifts given to him by God.

JOSEPH ANTHONY C. HERMOCILLA

ACKNOWLEDGEMENT

I would like to thank the following for their support and inspiration.

Members of my guidance committee Dr. Eliezier Albacea, Prof. Jaime M. Samaniego, and Prof. Concepcion L. Khan.

My other professors in graduate school, Prof. Elfredy V. Cadapan, Prof. Jaderick P. Pabico, Prof. Jeng Gendrano, Dr. Vladimir Mariano, Prof. Danilo Mercado, Dr. Santiago M. Alviar.

ICS and GS administrative staff, Kuya Mark, Mang Mandy, Tita Ides, Tita Viring, Tita Poi, Tita Oma, Kuya JJ, Kuya Reggie, Mang Bert, Sir Vernie, Tita Pam, Mang Niles, Kuya Alex, JunBac, Tita Pinky, Tita Connie, Tita Loti, Tita Cha, Tita Dorie

ICS faculty and colleagues, Arian, Dudai, Jas, JP, Dzeni, Liz, Tin De Sagun, Rizza, Grace, Anne, Jane, Rico, Leo, Rex, Mini, Ria, Val, Al, Tin Salaya, Ian, Duldz, Belvz, Paul, Makoy, Carla, Rev, TJ, Mykmyk, Mam Marge, Bob, Rainier, Sonny, Belvz, Sir Mics, Jo, Sir Ogot, Mam Janice, Sir Christian, Sir Wensley.

My former students and friends.

Ms. Phamela M. Papa and her family, Tito Ben, Tita Elvie, Ate Yza, Ate Ily, Nani, Tita Fe, Nay Mena.

My family and relatives, Mama, Daddy, Grace, Don, Warren, Mommy Paz, Marcel, Uncle Romy, Auntie Babes, Auntie Ching, Chino, Auntie Odie, Uncle Dong Dong, Auntie Leona, RV, RJ, Third.

The Lord for giving me the talents and gifts to help and serve others for His greater glory.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

HERMOCILLA, JOSEPH ANTHONY C., University of the Philippines Los Banos, December 2008. **A Multiagent System Framework for Solving the Student Sectioning Problem**

Major Professor: DR. ELIEZER A. ALBACEA

Assignment of students to classes is an important task in a university registration or enrollment process and is considered as a subproblem of the more general university timetabling problem which is NP-complete. This study characterizes the assignment of students to classes as the Student Sectioning Problem and proposes a multiagent system framework for solving it. The framework is composed of three types of agents namely scheduler agent, enlister agent, and student agent. The scheduler agent is responsible for creating the initial solution using iterative algorithms augmented with a maximum bipartite matching solver. Student agents communicate with enlister agents to improve the initial solution by performing cancellation and enlistment operations. A prototype system was implemented on top of the JADE multiagent platform using UPLB registration data. In general, the multiagent system framework provides a decentralized and concurrent approach to solving the Student Sectioning Problem and offers a more realistic model for the student registration process.

INTRODUCTION

Background of the Study

The University of the Philippines Los Banos (UPLB) is one of the biggest autonomous unit of the University of the Philippines System, offering several undergraduate and graduate degree courses. At present, there are approximately 10000 students both local and foreign, and about 78000 classes being offered every semester. A major task in UPLB, as with the case of any university, is the registration or enrollment process wherein students are assigned to classes they need to take in a semester or term to complete their degree.

In UPLB, before the end of the current semester, the University Registrar's Office collects a list of classes that each department will be offering for the incoming semester. The offering contains information on the classes that a department or institute will offer. These information include sections, time, day, instructor, room, and class size. The offering is generally a timetable and is constructed by the individual departments offering the subjects because they have more information and control on the resources. The departmental class offerings are then consolidated to produce a university class offering. The university class offering is then published.

Traditionally, the registrar's office manually assigns the students to classes to create the Form 5. This approach has been the practice for a long time but is ineffective, inefficient, and tedious to both administrators and students, especially with the increasing number of students each year. Using computers to do these tasks thus became a need.

In 1997, a set of computer programs, collectively called REGIST (Carino, 1997), was

developed using the FORTRAN programming language, to automate the registration process by addressing the problem of generating the write-in of each student(subjects they intend to take) and assigning students to classes in UPLB. Generating the write-in is important because it determines the subjects that a student will register. An incorrect set of subjects in the write-in can possibly delay a student from graduating on time. On the other hand, the problem of assigning students to classes is more important because the classes assigned to a student should not conflict in time and the classes should not exceed the number of allowed students.

Since the initial deployment of REGIST, no effort has been made to modify it, probably because it was developed using an old programming language and also because of the unavailability of detailed documentation and source code. However, several attempts have been made to develop algorithms to solve the same problems addressed by REGIST, particularly assigning students to classes. These algorithms are based on metaheuristics such as genetic algorithms(Marapao, 2004), tabu search(Guinto, 2004), and simulated annealing(Sandig, 2004).

More recently, System One (Duldulao, 2004) was developed to provide a web-based interface for the results generated by REGIST, specifically the Form 5. This allows students to view the classes in their Form 5 before the actual registration day. System One also provides a front-end for change of matriculation in case students want to add or remove classes. Since System One was developed using web-based technologies while REGIST using FORTRAN, the primary interface between the two systems is through text files.

Despite the attempts to develop new algorithms to solve the problem of assigning

students to classes and the creation of System One, REGIST remains to be the core of the registration process in UPLB. With the continuing changes in the academic guidelines, like the institution of RGEP, REGIST cannot easily accommodate them which essentially complicates the registration process.

In general, the registration process in a university can be viewed as a game with several players, each with a different set of objectives and desire to win. One player is the student which desires to obtain the required classes in order to graduate on time. Another player, the administration, desires to minimize the cost (number of faculty, classrooms, etc). The interaction between these players result to either achieving the objectives of the players or not. As with any game, some players will win and some players will lose.

The emerging field of multiagent systems, a subfield of artificial intelligence, can be used to model the registration or enrollment process in a university. A multiagent system is composed of individual autonomous agents that act on behalf of a user. The registration process when viewed as a game can be considered as a multiagent system with each player represented as an agent.

Statement of the Problem

The registration process is an important activity in a university. Its main objective is to assign students to classes. Given that the classes are often limited by the number of students that can enroll, not all students can be assigned to classes. Evidently, automating this task requires careful characterization of the problem in order to properly validate the results. Thus this study will characterize the problem of assigning students to classes as the

Student Sectioning Problem and propose an algorithm to solve the problem in the context of multiagent systems.

Significance of the Study

This study will add to the current approaches to improve the automated registration process by characterizing the assignment of students to classes as the Student Sectioning Problem. This characterization will allow validation of results and at the same time provide a basis of comparison among the current approaches to solve the problem.

Multiagent systems is currently an active area of research because of the ubiquity of the network and the move by large enterprises to decentralize the deployment of applications. Also, certain problem domains can be best characterized and solved using a multiagent systems framework, especially domains which require cooperation and negotiation.

Objectives of the Study

The aim of this study is to characterize the assignment of students to classes as the Student Sectioning Problem and propose algorithms to solve this problem in the context of multiagent systems. The objectives are the following:

1. Develop an Application Programming Interface (API) to model the Student Sectioning Problem;
2. Define quantitative measures and domain specific heuristics to evaluate the quality of a solution to the Student Sectioning Problem;

3. Compare the solution produced by the proposed algorithms to the solution produced by REGIST using UPLB registration data;

4. Develop a prototype multiagent system that solves the Student Sectioning Problem.

REVIEW OF RELATED LITERATURE

Universities offering a large number of classes to a large number of students need to automate the registration process in order to make it more efficient. Thus, the characterization of problems related to the registration process and finding solutions to these problems have been the subject of interest to researchers in various fields. A central problem related to the registration process that has been formalized is called the timetabling problem. In general, the timetabling problem consists of a set of work periods, a set of craftsmen, a set of tasks, a subset of available hours for each craftsman, a subset of available hours for each task, and for each pair of craftsman and task, a number of required work periods (Garey, 1979). The goal is to find a set of assignments of craftsman to task that satisfies certain constraints. The timetabling problem is an NP-complete problem and thus there is no known polynomial time algorithm for solving it using deterministic algorithms. Most approaches to solving timetabling problems use variants of local search and metaheuristics. The international conferences on the Practice and Theory of Automated Timetabling (PATAT) has been successful in providing a venue for researchers to share their work related to the timetabling problem.

In a university setting, the timetable generation, aside from being inherently hard to solve based on complexity theory, is even harder to manage manually considering a large number of resources (or tasks) and people involved. Also, in a typical university, individual classes are offered by separate departments, making it hard to come up with a global view. As a result, general timetabling is not solved as is, but rather is divided into phases. In the case of UPLB, the departments and institute create their respective class offerings and

submit them for consolidation to the registrar's office. The assignment of students to classes is performed manually to create the students' schedules. Using this phased-approach, the challenge becomes finding a good assignment of students to classes. The following sections summarizes the current approaches to solve this challenge in the UPLB setting.

Greedy Algorithm

REGIST

REGIST(Carino, 1997) is the first attempt in UPLB to automate the registration or enrollment process. It was developed using the FORTRAN programming language. Two main problems are addressed by REGIST namely generating the write-in of students and assigning students to classes. The first problem is solved by considering the curriculum of a student as a task network. The curriculum describes the list of subjects to take to finish a degree. Using the Critical Path Method (CPM), a set of subjects to take is generated that minimizes the number of semesters of stay of a student in the university. The assignment of students to sections is solved using a best first search algorithm(Carino, 2007).

Metaheuristics

Genetic Algorithm

In 2004, Marapao developed a genetic algorithm to solve the assignment of students to classes by formulating the problem as a timetabling problem. The average percentage load

was defined as the objective function to maximize. The initial solution used to seed the algorithm is the solution generated by REGIST.

Simulated Annealing

In 2004, Sandig developed an algorithm to solve the student sectioning problem based on simulated annealing. By treating the problem as a timetabling problem and using the average percentage load as the objective function, the algorithm attempted to improve the solution generated by REGIST.

Tabu Search

In 2004, Guinto developed a tabu search based algorithm to solve the student sectioning problem. Again the problem was treated as timetabling problem with the average percentage load as the objective function to optimize. Initial solution used was the solution produced by REGIST.

THEORETICAL FRAMEWORK

Student Sectioning Problem Definition

We formally describe the Student Sectioning Problem(SSP) as a tuple, $SSP=(A,B,C,D)$ where A is a set of students, B is a set of subjects, C is a set of classes with elements $c=(b,e)$, where $b\in B$ and e is a section. We specify additional attributes to a class c such as *timeslot*. We define *slot* as a pair $t=(c,n)$, where c is a class, and n is the *slot number*. We let E be the set of all slots. The $classsize(c)$ is the number of slots with c in the elements of E. D is the set of write-in with elements also a pair $d=(a,b)$ where $a\in A$ and $b\in B$. We define an *assignment* as a pair $f=(d,t)$ such that given a write-in $d=(a,b)$ and the slot set E, $(b,e)\in C$ and c is in $t\in T$. We also define a predicate $conflict(f1,f2)$ over a set of assignments Q such that given any two assignments *f1* and *f2* in Q, it returns true if the *timeslots* of the c in *f1* and c in *f2* are the same, false otherwise. In addition, we define another predicate $full(c)$ over a set of assignments Q such that it returns true if the number of assignments in Q which include c is greater than $classsize(c)$, false otherwise. The solution to a SSP is a set S of assignments such that for all students a in A, the subset X_a of S containing all assignments for student a, $conflict(f_{1a},f_{2a})$ is false, and for all c in S, $full(c)$ is false. We refer to X_a as the *schedule* of student a. The union of all X_a for all a in A is

the set S . The example below illustrates an instance of the problem and a possible solution.

$$A=\{1996-47938, 2001-56024\}$$
$$B=\{CMSC\,128, CMSC\,125\}$$
$$C=\{(CMSC\,128, AB), (CMSC\,128, C), (CMSC\,125, C)\}$$
$$D=\{\begin{matrix}(1996-47938, CMSC\,128), (1996-47938, CMSC\,125)\\ (2001-56024, CMSC\,128), (2001-56024, CMSC\,125)\end{matrix}\}$$
$$E=\{((CMSC\,128, AB), 1), ((CMSC\,128, C), 1), ((CMSC\,125, C), 1)\}$$

$$S=\{\begin{matrix}((1996-47938, CMSC\,128), ((CMSC\,128, AB), 1)),\\ ((1996-47938, CMSC\,125), ((CMSC\,125, C), 1)),\\ ((2001-56024, CMSC\,128), ((CMSC\,128, C), 1))\end{matrix}\}$$

$$X_{1996\text{-}47938}=\{\begin{matrix}((1996-47938, CMSC\,128), ((CMSC\,128, AB), 1)),\\ ((1996-47938, CMSC\,125), ((CMSC\,125, C), 1))\end{matrix}\}$$
$$X_{2001\text{-}56024}=\{((2001-56024, CMSC\,128), ((CMSC\,128, C), 1))\}$$

An alternative formulation of the Student Sectioning Problem can be made. This is accomplished by redefining assignment. We redefine *assignment* as a pair $g=(t, d)$ where t is an element of E and d is an element of D . This formulation makes it easy to introduce heuristics during the solving process compared to the original formulation of the problem.

$$R=\{\begin{matrix}(((CMSC\,128, AB), 1), (1996-47938, CMSC\,128)),\\ (((CMSC\,125, C), 1), (1996-47938, CMSC\,125)),\\ (((CMSC\,128, C), 1), (2001-56024, CMSC\,128))\end{matrix}\}$$

Constraint Satisfaction Problems

Russel and Norvig (Russell, 2003) defined a Constraint Satisfaction Problem(CSP) in the following manner.

> "..a set of *variables* X_1, X_2,...,X_n and a set of *constraints* C_1,C_2,...,C_m. Each variable X_i has a nonempty *domain* D_i of possible *values*. Each constraint C_i involves some subset of the variables and specifies the allowable combination of values for that subset. A *state* of the problem is defined by an *assignment* of values to some or all of the variables, $\{X_i=v_i, X_j=v_j,...\}$. An assignment that does not violate any constraints is called a *consistent* or *legal* assignment. A *complete* assignment is one in which every variable is mentioned, and a *solution* to a CSP is a complete assignment that satisfies all the constraints.."

We can formulate a Student Sectioning Problem, $SSP=(A, B, C, D)$, as a $CSP=(V, U, W)$ by mapping each element of D (or E in the alternative formulation) to the set of variables V, the set $\{conflict(f1, f2), full(c)\}$ as the set of constraints W, and the domain set U as set of the subsets of the power set of E. An assignment in SSP directly corresponds with an assignment in CSP. Although a solution in a CSP must be a complete assignment, according to the above definition, it may not be the case for a SSP because of resource constraints, such as a limited number of slots in classes. SSP can be considered as an over-constrained CSP. A partial assignment therefore is acceptable as a solution to a SSP as long as no constraints are violated. CSPs are a well-studied problems in artificial intelligence and are usually solved via search methods. *Backtracking*, *constraint propagation*, and *local search* are specific approaches used to solve CSPs(Russell, 2003). Thus, given the mapping specified above, we can solve the SSP using algorithms for solving CSPs.

A CSP can also be converted to an optimization problem by defining a function that maps a given solution A to a cost, $f: A \rightarrow \mathbb{R}$. The most basic function is one which

counts the number of violated constraints. A CSP then can be viewed as a minimization problem that minimizes the number of violated constraints, assuming a complete assignment.

One systematic approach to solving CSPs is backtracking. Backtracking is similar to a depth-first search which chooses values for one variable at a time and backtracks when a variable has no legal values left to assign (Russell, 2003). The pseudocode for backtracking adapted from Russell is given below.

```
function BACKTRACKING(csp)
  return RECURSIVE-BACKTRACKING({}. csp)

function RECURSIVE-BACKTRACKING(assignment, csp)
  if assignment is complete return assignment
  var <-- SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment,
csp)
  foreach value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment according to CONSTRAINTS[csp]
      add { var = value } to assignment
      result <-- RECURSIVE-BACKTRACKING(assignment, csp)
      if result <> failure return result
      remove { var = value } from assignment
  return failure
```

Figure 1. Recursive backtracking for solving CSP

Backtracking is generally an uninformed algorithm which can be very ineffective for large problems because of the large search space. It can further be improved by using heuristics. One such heuristic is specifying a particular ordering in the variables to assign, as well as in the values. As an example, the *Minimum Remaining Values (MRV)* (Russell,

2003) heuristic chooses the variable with the fewest legal values, allowing early detection of inconsistent assignments. Another approach to improve the backtracking technique is by *forward checking*. When a variable X is assigned, it examines all the unassigned variables Y connected to X by a constraint and deletes from the domain of Y any value that is inconsistent with the value chosen for X . *Constraint propagation* is yet another technique for early detection of inconsistencies. Constraints on one variable are propagated to other variables before or during the search process. A constraint graph is used to represent constraints in a CSP. Variables represent nodes and an edge between two nodes exists if there is a constraint between the variables represented by the nodes.

Backtracking techniques in general are not memory efficient because the path to a solution is stored. *Local search* methods are techniques for solving problems in which paths to the solution is not important, only the final configuration or solution. Basic search methods are examples of techniques that take note of the paths that lead to the goal or solution. Local search for CSPs uses a *complete-state formulation* which initially assigns values to *all* the variables. A *successor* function changes the value of one variable at a time. The value selected for a variable is the one which results in the minimum number of conflicts with other variables and is referred to as the *min-conflicts heuristics(*Russell, 2003). The pseudocode below illustrates the min-conflicts heuristics local search for CSPs.

```
function MIN-CONFLICTS(csp, max_steps)
  current <-- an initial complete assignment for csp
  for i = 1 to max_steps do
    if current is a solution for csp return current
    var <-- a randomly chosen, conflicted variable from VARIABLES[csp]
    value <-- the value v for var that minimizes CONFLICT(var, v, current, csp)
```

```
    set var = value in current
  return failure
```

Figure 2. The min-conflict heuristic local search technique for CSP

Agents

An agent can be defined as "a computer system capable of autonomous action in some environment" (Wooldridge, 2002). An example of an agent is a thermostat or a Unix daemon. A more interesting kind of agents are called intelligent agents. Intelligent agents are *reactive*, *proactive*, and *social*. Being reactive, intelligent agents continuously interact with its environment. An agent perceives its environment through *sensors* and acts on it through *actuators*. A *proactive* agent is capable of generating and attempting to achieve goals and thus exhibit goal directed behavior. The *social* aspect of an intelligent agent is its ability to interact with other agents in the environment. The interaction may be *cooperation*, *competition*, or *negotiation*. Other properties such as *mobility*, *veracity*, *benevolence*, *rationality*, and *learning* are often discussed when reasoning about agents.

The agent's perceptual inputs is referred to as *percepts* and the *percept sequence* is the history of everything the agent has ever perceived. The agent's actions at any given time can depend on the percept sequence. An agent's behavior can then be defined by an *agent function* that maps the percept sequence to action (Russell, 2003).

Rationality of agents refers to the ability of the agent to do the *right* thing. The *right* thing can mean that an agent is successful in accomplishing its goal. A performance measure is thus required in order to assess the success of an agent's behavior. A rational

agent can be defined in such a way that for each possible percept sequence, the agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has. (Russell, 2008).

Multiagent Systems

Multiagent systems (MAS) are computer systems wherein several agents, possibly heterogeneous, are interacting to solve a problem that an individual agent cannot solve. MAS research is a subfield of Distributed Artificial Intelligence together with Distributed Problem Solving (DPS). The main concern of MAS research is the behavior of a collection of autonomous agents in order to solve a given problem. Foreseen advantages of MAS compared to single, monolithic, and centralized problem solvers includes faster problem solving, decreased communication, more flexibility, and increased reliability (Wooldridge, 2002).

Belief-Desire-Intention Architectures

Models provide formalisms to describe characteristics of a multiagent system. The Belief-Desire-Intention or BDI model is by far the most studied and applied model of agency. In this model, an agent is regarded as an *intentional system*, such that an agent has mental attitudes and these attitudes define the behavior of a rational agent. Conventional computer systems are usually algorithmic and task-oriented with complete access to the

information needed for computations. However, in computer systems that are dynamic, uncertain, resource-bounded, and with a limited view of the environment, conventional architecture is not the best solution.

In systems when an agent needs to act, the agent needs to select appropriate actions from a given set of options. The selection function needs at least two inputs to determine the appropriate action. The first is that the agent needs to have a knowledge of the state of the environment. These are called *beliefs*. Beliefs are represented as state variables. Second, an agent has to have information regarding the payoffs of its objectives, the performance measure which is represented functionally rather than state variables. These are referred to as *desires*. Given these two inputs, decision-theory can be used to generate the appropriate action to take. However, since the system is dynamic, changes may occur. The change may occur while evaluating the selection function or during the execution of the action itself. In the first case, a faster selection function can be employed. In the second case, when using decision theory, the selection function would need to be reevaluated in order to incorporate the most recent changes. However, this operation is too expensive. A better approach would be to store the state that represents the most recently chosen course of action. This will allow an agent to somehow instantaneously determine potentially significant changes. This component is referred to as *intentions*. Practical reasoning agents has these characteristics of beliefs, desires, and intentions. Formal representations of the BDI architecture have also been developed to provide a theoretical framework when reasoning about multiagent systems based on BDI architecture (Rao, 1995).

Negotiation and the Contract Net Protocol

In multiagent systems, interaction among agents may be cooperation, competition, or negotiation. Negotiation as defined by Smith and Davis is an "organizing principle."

By negotiation, we mean a discussion in which interested parties exchange information and come to an agreement. For our purposes, negotiation has three important components: (a) there is a two-way exchange of information, (b) each party to the negotiation evaluates the information from its own perspective, (c) final agreement is achieved by mutual selection.

In general, negotiation is used for task and resource allocation. The Contract Net Protocol (CNP) was introduced by Smith in 1980 to handle such tasks. It works using the following sequence of events:

1. *Recognition* - an agent recognizes that it needs help
2. *Announcement* - the agent announces the task specification to other agents
3. *Bidding* - agents who receive the announcement decide whether to bid. Agents who decide to bid must submit a tender
4. *Awarding* - agent who announced will select the agent to award the contract
5. *Expediting* - successful contractor expedites the task

Foundation of Intelligent Physical Agents

FIPA "was originally formed as a Swiss based organization in 1996 to produce software standards specifications for heterogeneous and interacting agents and agent based systems. Since its foundations, FIPA has played a crucial role in the development of agents standards and has promoted a number of initiatives and events that contributed to

the development and uptake of agent technology. Furthermore, many of the ideas originated and developed in FIPA are now coming into sharp focus in new generations of Web/Internet technology and related specifications."

FIPA, the standards organization for agents and multi-agent systems was officially accepted by the IEEE as its eleventh standards committee on June 8, 2005. Thus, being a standard, improvements in the specifications is expected to come out. Several implementations of the FIPA specifications are available. One is an implementation from Tilab which is called the Java Agent Development Framework (JADE). An extension of JADE, JADEX implements the Belief-Desire-Intention architecture and treats each component (belief, desire, intentions) as first class objects.

Distributed Constraint Satisfaction Problems

A Distributed Constraint Satisfaction Problem(DisCSP) is a Constraint Satisfaction Problem in which variables and constraints are distributed among agents(Yokoo, 1998). DisCSP relies on a communication model that assumes that agents communicate by sending messages and the delay in delivering a message is finite. Formally, DisCSP is composed of m agents $1, 2, 3, \ldots, m$. Each variable x_j belongs to one agent i (represented as a relation $belongs(x_j, i)$). Constraints are also distributed among agents such that given a constraint predicate p_k known to an agent l , it is represented as $known(p_k, l)$. A DisCSP is solved if all the variables are assigned values that satisfies all constraint predicates.

A trivial algorithm to solve a DisCSP is for the agents to elect a leader agent and send

all variables and constraints to the leader agent. The leader agent then solves DisCSP just like an ordinary centralized CSP. Another algorithm, called synchronous backtracking, specifies an ordering of the agents and each agent assigns values to their variables one by one. The partial assignment is then passed on to the next agent. If an agent cannot assign values its variables, it sends a backtrack message to the agent where it received the partial assignment which performs a reassignment. These algorithms however fail to take advantage of concurrency. Other algorithms for solving DisCSP are Asychronous Backtracking(Yokoo, 1980), Asynchronous Weak-commitment Search, and ADOPT(Modi, 2001) which are fully distributed and concurrent.

Maximum Bipartite Matching Problem

Given a bipartite graph $G=(V=(X,Y),E)$, we define a *bipartite matching* as a set of edges M in G such that no two edges in M share a common vertex. A vertex is matched if it is incident to an edge in M. The *maximum bipartite matching problem* is to find bipartite matching which contains the largest possible number of edges. A maximum bipartite matching problem can be solved using the augmenting path algorithm. This solution is equivalent to finding a *maximal flow* in a network by converting G into a network $N=(source, sink, G)$. This is done by adding a super source vertex in G with edges to all vertices in X and a super sink vertex with edges from all the vertices in Y. All edges with flow from X to Y constitute a maximum matching(Cormen, 2001).

METHODOLOGY

Data Collection and Filtering

Data used and generated by REGIST were collected and filtered. The data collected includes *Write-in*, *Class Offering*, and *Form 5*. These data are available as comma-separated values (CSV) text files obtained from the registrar's office .

```
CMSC 1,YZ,80,5-6,TTh,ICS LH1,TBA,TBA
CMSC 1,YZ-1L,15,10-1,Mon,ICS PC Lab 1,TBA,TBA
CMSC 1,YZ-2L,15,10-1,Tues,ICS PC Lab 1,TBA,TBA
CMSC 1,YZ-3L,15,10-1,Wed,ICS PC Lab 1,TBA,TBA
CMSC 1,YZ-4L,15,10-1,Thurs,ICS PC Lab 1,TBA,TBA
CMSC 1,YZ-5L,20,1-4,Thurs,ICS PC Lab 4,TBA,TBA
CMSC 100,C,80,9-10,MW,ICS LH3,TBA,TBA
CMSC 100,C-1L,20,1-4,Mon,ICS PC Lab 6,TBA,TBA
CMSC 100,C-2L,20,1-4,Tues,ICS PC Lab 6,TBA,TBA
CMSC 100,C-3L,20,1-4,Wed,ICS PC Lab 6,TBA,TBA
CMSC 100,C-4L,20,1-4,Thurs,ICS PC Lab 6,TBA,TBA
```

Figure 3. Sample class offering data

Each line is a class entry composed of the *subject*, *section name*, *class size*, *time*, *days*, *room*, and *instructor*. TBA is means To be Assigned. Section names can either be for lecture or laboratory. Those suffixed with *-xL*, where *x* is a number, indicates laboratory section names for the given lecture. For example *YZ* is the lecture section name for *CMSC 1*and *YZ-1L* is the section name of the first laboratory class for *CMSC 1*.

```
2001-56228,2004,SECOND,CMSC 1,,3,REGD,1,3,2
2004-40409,2004,SECOND,CMSC 2,,3,REGD,1,3,4
2001-01179,2004,SECOND,CMSC 128,,3,REGD,1,9,4
2001-01179,2004,SECOND,CMSC 125,,3,REGD,2,9,4
2001-01179,2004,SECOND,CMSC 142,,3,REGD,3,9,4
2004-14195,2004,SECOND,CMSC 2,,3,REGD,1,3,4
2000-47041,2004,SECOND,CMSC 142,,3,REGD,1,5,2
2000-47041,2004,SECOND,CMSC 190-2,,2,REGD,2,5,2
2004-28322,2004,SECOND,CMSC 2,,3,REGD,1,3,4
2002-17224,2004,SECOND,CMSC 1,,3,REGD,1,3,4
```

Figure 4. Sample write-in data

Each line is a write-in entry composed of the *student number*, *year*, *semester*, *subject*, *section name*, *units*, *grade*, *courserank*, *allowed number of units*, *student priority*. A REGD value for grade means subject is still to be taken. The course rank refers to the importance of the subject to the student, a low value indicates that it is a priority subject. The student priority is described below.

Table 1. Priority ordering

PRIORITY	DESCRIPTION
1	New Student (no record of any grade)
2	Graduating (24 units or less to earn in curriculum)
3	New Student (no record of any grade)
4	Failed (0-50%] of units last sem (status code 2 or 3)
5	Failed (50%,75%] of units, or on LOA last sem
6	Failed (75%-100%] of units last sem (status code 5 or 6)

```
2000-13610,2006,SECOND,PE 2,BT-B,0,REGD,1,21,3
2000-13610,2006,SECOND,CHEM 40,V-1L,4,REGD,2,21,3
2000-13610,2006,SECOND,CHEM 32,U-1L,5,REGD,3,21,3
2000-13610,2006,SECOND,MGT 111,KL,3,REGD,4,21,3
2000-13610,2006,SECOND,AGRI 199,J2,1,REGD,5,21,3
2000-13610,2006,SECOND,FST 170,D,3,REGD,6,21,3
2000-13610,2006,SECOND,FST 199,J2,1,REGD,7,21,3
2000-13610,2006,SECOND,STS 1(MST),A,3,REGD,8,21,3
2000-13610,2006,SECOND,FST 147,,3,REGD,9,21,3
```

Figure 5. Sample Form5 data

Each line consists of student number, year, semester, subject, section name, units, grade, index, allowed number of units, and student priority. The format for each entry is almost the same as the entry for write-in except that the section name field has an entry.

API Design

An Application Programming Interface (API) was developed using the Java programming language to allow interoperability with external systems, such as System One which was also developed using Java. The API consists of a collection of classes and interfaces to model entities and processes in the registration. The API will be used in the implementation of the algorithm for solving the Student Sectioning Problem. The following are some of the essential classes and interfaces in the API.

WriteIn

Encapsulates student write-in information. This class contains methods such as for retrieving subjects in the write-in, determining the priority of the student, and getting the allowed units for the student.

Section

Contains methods for obtaining class information such as the day, time, room, instructor, class size, and time slot.

Timeslot

Assigns an integral value to a (day, time) pair to represent a time slot that is used by a class. For example, the value 0 represents (Monday, 7-8) and the value 1 represents (Monday, 8-9).

DefaultForm5

Represents a student's Form 5 which contains the classes to take. This class contains methods for getting the section assignments, getting assigned number of units, among others.

Classlist

Contains a list of students enlisted to a class.

IOffering

An interface that encapsulates class offering. This interface defines methods for obtaining the lecture and laboratory sections for a given subject.

Solution Quality Criteria

A set of criteria or measures were defined to evaluate the quality of a solution to the Student Sectioning Problem. A solution to the Student Sectioning Problem is a set of assignments S such that for all X_a as a subset, there is no conflict in a 's schedule and for all assignments in S and no class contains a number of students that exceed the class size. Given a $SSP=(A, B, C, D)$, we let D_a (write-in of student a) be the subset of D with elements containing a as entry. We define $unitsAllowed(D_a)$ over D as the number of credit units allowed in the write-in and $unitsAssigned(X_a)$ over S as the number of credit units in the schedule of a . Given c element of C , we also define $assigned(c)$ over S as the number of students assigned to class c . Given a subset D_b of D such that b is an entry in an element of D , let $demand(b)$ be the number of elements in Db . Let $slots(b)$ be the sum of the class sizes of all classes in C such that b is an entry in an element of C. We define the following measures for describing and evaluating the solution to a SSP.

- $Number\ of\ Assignments(NA)$
  - Given a $SSP=(A, B, C, D)$, we want a solution with the maximum

number of assignments possible, $max(|S|)$

- *Number of Students with Full Load* $(NSFL)$
  - $NSFL = |\{a \in A \mid unitsAllowed(D_a) = unitsAssigned(X_a)\}|$
- *Number of Students Underloaded* (NSU)
  - $NSU = |\{a \in A \mid unitsAllowed(D_a) > unitsAssigned(X_a)\}|$
- *Number of Students Overloaded* (NSO)
  - $NSO = |\{a \in A \mid unitsAllowed(D_a) < unitsAssigned(X_a)\}|$
- *Number of Students with Zero Load* $(NSZL)$
  - $NSZL = |\{a \in A \mid unitsAssigned(X_a) = 0\}|$
- *Average Percentage Load* (APL)
  - Let *Percentage Load* (*PL*) of student *a* be

$$PL_a = \{\frac{unitsAssigned(X_a)}{unitsAllowed(D_a)}\} \times 100$$. Then, $$APL = \frac{\sum_{a=1}^{|S|} PL_a}{|S|}$$.

- *Average Percentage Class Size (APCS)*
  - Let *Percentage Class Size (PCS)* of *c*

$$PCS_c = \{\frac{assigned(c)}{classsize(c)}\} \times 100$$. Then, $$APCS = \frac{\sum_{c=1}^{|C|} PCS_c}{|C|}$$.

- *Demand-Slots Ratio (DSR)*
  - $DSR_b = \frac{demand(b)}{slots(b)}$

Algorithm Design and Implementation

Iterative Algorithm (IA)

Given a $SSP = (A, B, C, D)$, the algorithm orders B , the set of subjects, based on the $demand - slots\,ratio$. Slots in subjects with low demand-slots ratio are assigned first. Majority of subjects with low demand-slots ratio are major subjects and those with high demand-slots ratio are usually service subjects . Using this heuristic, we are trying to guarantee that we assign major subject slots first and service subjects last. The function RANK-SUBJECTS() implements this heuristic and returns a . Subjects of the same rank are then shuffled. The algorithm iterates over each subject and obtains a list of students demanding the current subject being processed. The students are ranked based on three criteria: *percentage number of available section options*, *current percentage load*, and *priority of subject* in the students write-in. The overall rank of the student is determined by the weighted sum of the mentioned criteria. The function RANK-STUDENTS() implements this. For each student of the same rank, the algorithm looks for a class with available slots that does not conflict with the current student's schedule. This step is implemented in the function ENLIST(). The selection of class is done randomly, by shuffling the list of classes that does not conflict with a students current schedule.

```
function IA(A, B, C, D) returns a set of SCHEDULE
  subjects <-- RANK-SUBJECTS(B)
  foreach sub in subjects do
    students <-- RANK-STUDENTS(DEMAND(D,sub))
    foreach stud in students do
      ENLIST(A,B,C,D, stud, sub)

function ENLIST(A, B, C, D,student, subject)
  classes <-- RANK-CLASSES(C,subject)
  foreach class in classes do
    if not CONFLICT(SCHEDULE(student), class)
      slot <-- AVAILABLE-SLOT(class)
      if slot not empty
        add slot to SCHEDULE(student)
        break
```

Figure 6. The IA algorithm for solving SSP.

Iterative Algorithm with Maximum Bipartite Matching (IAMBM)

The iterative algorithm above was augmented with a Maximum Bipartite Matching(MBM) solver. An instance of $MBM_b, G=(V=(L,R),E)$ for subject b is created by letting the set L contain the students demanding subject b and the set R contain the slots for all classes of b. An edge is added between student l and slot r if the schedule of l does not conflict with the class for r. To solve the MBM instance, G is converted to a network $N=(source, sink, G)$ by adding a *super source vertex* and a *super sink vertex*. Edmonds-Karp max-flow algorithm was used to find the flows in the network which represent the matching. The solver was implemented in the function EDMONDS-KARP-MAX-FLOW() (Cormen, 2001).

```
function IAMBM(A, B, C, D) returns a set of SCHEDULE
  subjects <-- RANK-SUBJECTS(B)
  foreach sub in subjects do
    L <-- DEMAND(D,sub))
    R <-- SLOTS(sub)
    G <-- graph with vertex sets L and R
    foreach student in L do
      foreach slot in R do
        if not CONFLICT(SCHEDULE(student), CLASS(slot))
          add edge (student, slot) to G
    N <-- network (source, sink, G)
    M <-- EDMONDS-KARP-MAX-FLOW(N)
    foreach (student, slot) in M do
      add slot to SCHEDULE(student)
```

Figure 7. The IAMBM algorithm for solving a SSP.

Multiagent System Framework Design and Implementation

We look at the registration process as being composed of several actors, each with a particular role to play in the process. The first actor is the *registrar* whose main function is the overall management of the registration process. Second, is a department's *academic committee head* who creates the department's class offering. The third actor is the *enlister* who accepts enlistment and *c*ancellation requests. Lastly, the *student* who enlists or cancels classes. A typical registration process scenario can have several instances of these actors interacting.

A prototype multiagent system was developed using the Java Agent Development Environment (JADE). The multiagent is composed of a scheduler agent, enlister agents, and student agents.

The scheduler agent, which act the role of the registrar, is the main agent that initializes the simulation. It has access to the students, subjects, classes and write-in data needed for solving the SSP. The scheduler agent instantiates a set of enlister agents for each subject. Since the scheduler agent has all the necessary inputs for solving the SSP, it can generate an initial solution using the algorithms described above, IA and IAMBM.

An enlister agent requests for the classlists for all the sections in the subject assigned to it from the scheduler agent. The enlister agents then waits for enlistment or cancellation requests from student agents.

Student agents are also instantiated by the scheduler agent. Once created, a student agent requests for the the Form 5(schedule) from the scheduler agent. The schedule may contain initial assignments if the scheduler agent generated an initial solution. The student agent evaluates the contents of the Form 5 to determine the next action to take, whether to cancel or enlist.

The multiagent system framework solves the Student Sectioning Problem in a similar fashion that agents solves a DisCSP. The main solvers are the student agents. Each agent holds a set of *variables* (write-in) that must be assigned to values (slots). A *happy* state is defined for each student agent which means that it has a complete assignment or is satisfied with the current assignment (*desire* in the BDI agent architecture) of subject to sections. If after some number of attempts and the student agents fails to obtain a complete assignment, it discards all previous assignments but saves *one assignment* for itself so that it will not have a zero assignment. This approach minimizes the number of agents with zero assignments (NSZL). The enlister agents act as constraint checker agents by ensuring that only in classes with available are students allowed to enlist (domain filtering in CSP). After

a few number of failed attempts to complete its assignment, the student agent submits its final assignment to the scheduler agent which finalizes and commits it.

RESULTS AND DISCUSSION

The algorithms described in the methodology were implemented in the Java programming language. Experiments were conducted to evaluate the results using the data obtained. Statistics of solutions were collected using sample data from Second Semester 2004. The programs were run in an IBM Thinkpad R50e with 1GB RAM

Table 2.Statistics for solutions generated by the algorithms using 2nd Sem 2004.

ALGO	WRITE-IN	NO WRITE-IN	APL	NSFL	NSU	NSO	NSZL	APCS	GTE15	LT15
REGIST	8603	362	88.18	3540	4089	908	66	72.91	6008	2595
IA	8603	0	86.66	3364	5193	0	46	71.20	6008	2595
IAMBM	8603	0	87.57	3533	4986	33	51	71.80	6164	2439

The table above shows that REGIST has the highest APL of 88.18. Also, REGIST allows overloading which result to 908 students being overloaded. It is possible that allowing overloading can increase the APL. Allowing overloading however deprives other students from obtaining classes. The column GTE15 are the number of students who obtained *greater than or equal* to 15 units. This measure is important in the case of UPLB because it acts as a threshold for having a regular load. The column LT15 refers to the number of students who obtained *less than fifteen* units.

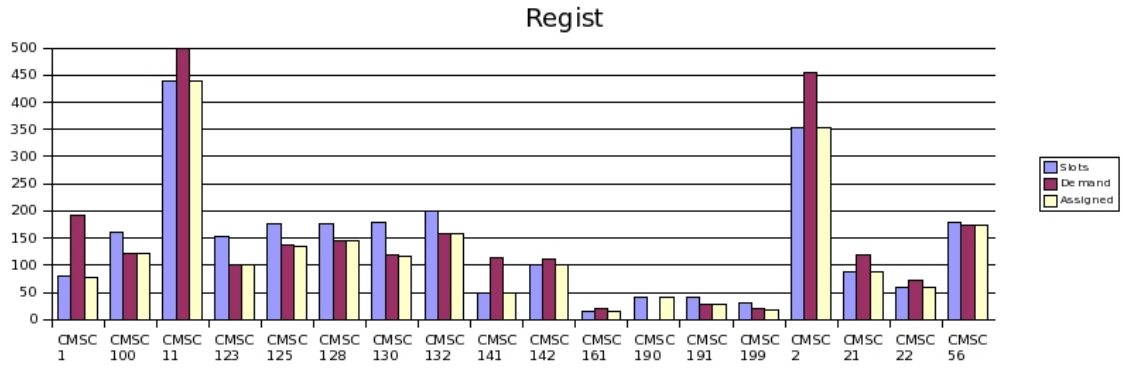

Figure 8. Available slots, demand, and assigned slots for CMSC subjects using REGIST

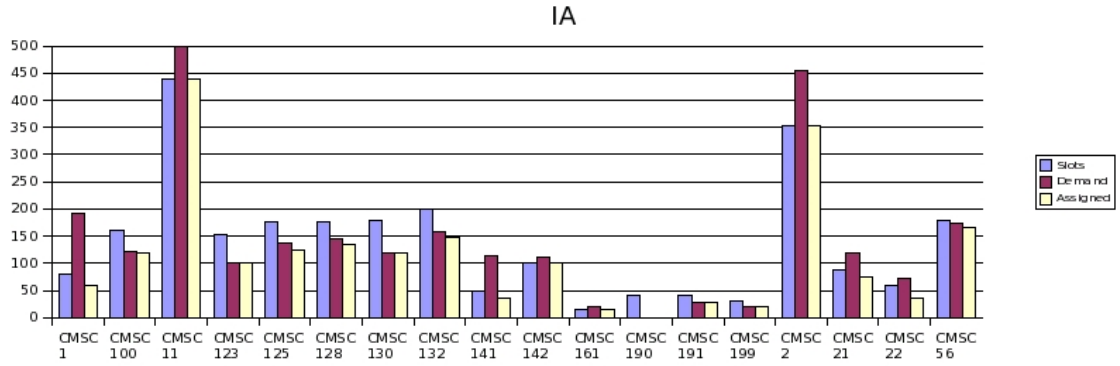

Figure 9. Available slots, demand, and assigned slots for CMSC subjects using IA

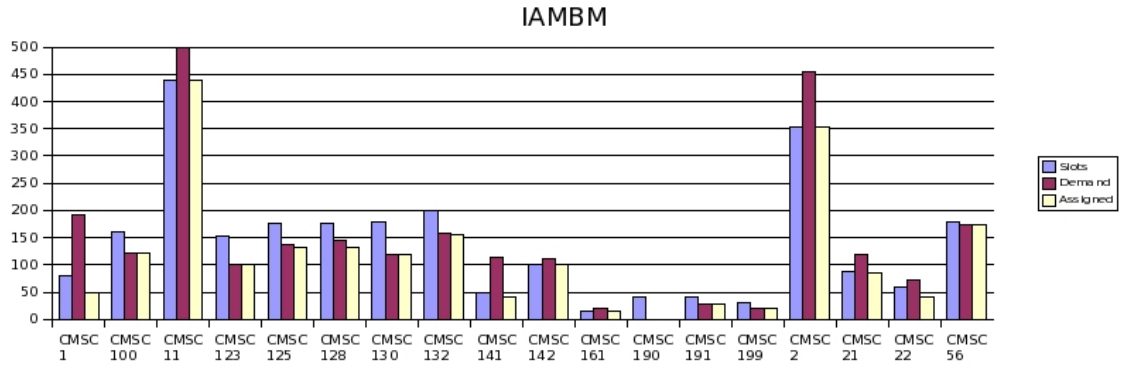

Figure 10. Available slots, demand, and assigned slots for CMSC subjects using IAMBM

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
7-8	NASC 3(MST) A		NASC 3(MST) A		NASC 3(MST) A	
8-9	CMSC 130 B	CMSC 22 ST	CMSC 130 B	CMSC 22 ST		
9-10		CMSC 123 T		CMSC 123 T		
10-11		CMSC 22 ST-2L		CMSC 130 B-3L		
11-12		CMSC 22 ST-2L		CMSC 130 B-3L		
12-1	STAT 101 F	CMSC 22 ST-2L	STAT 101 F	CMSC 130 B-3L		
1-2						
2-3						
3-4						
4-5			CMSC 123 T-3L	STAT 101 F-6L		
5-6			CMSC 123 T-3L	STAT 101 F-6L		
6-7			CMSC 123 T-3L	STAT 101 F-6L		

Figure 11. Sample schedule generated using REGIST

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
7-8						
8-9	STAT 101 B	CMSC 130 ST	STAT 101 B	CMSC 130 ST		
9-10		CMSC 123 T		CMSC 123 T		
10-11	STAT 101 B-2L		CMSC 123 T-2L			
11-12	STAT 101 B-2L		CMSC 123 T-2L			
12-1	STAT 101 B-2L		CMSC 123 T-2L			
1-2		PSY 1(SSP) W1		PSY 1(SSP) W1		
2-3		PSY 1(SSP) W1		PSY 1(SSP) W1		
3-4	HUM 2(AH) I2		HUM 2(AH) I2		HUM 2(AH) I2	
4-5	CMSC 130 ST-1L					
5-6	CMSC 130 ST-1L					
6-7	CMSC 130 ST-1L					

Figure 12. Sample schedule generated using IA

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
7-8						
8-9	STAT 101 B	CMSC 130 ST	STAT 101 B	CMSC 130 ST		
9-10		CMSC 123 T		CMSC 123 T		
10-11	CMSC 123 T-1L		STAT 101 B-5L			
11-12	CMSC 123 T-1L		STAT 101 B-5L			
12-1	CMSC 123 T-1L		STAT 101 B-5L			
1-2		PSY 1(SSP) W2		PSY 1(SSP) W2		
2-3	NASC 3(MST) H	PSY 1(SSP) W2	NASC 3(MST) H	PSY 1(SSP) W2	NASC 3(MST) H	
3-4						
4-5				CMSC 130 ST-3L		
5-6				CMSC 130 ST-3L		
6-7				CMSC 130 ST-3L		

Figure 13. Sample schedule generated using IAMBM

By filtering only CMSC (Computer Science subjects) to be included in the class offering and at the same time filtering the write-in with only those for CMSC subjects, the following results were obtained. The reason for filtering the data is because JADE fails to create threads when instantiating all 8603 student agents. Ideally, a MAS is run over multiple nodes but it was not considered in this study.

Table 3: Statistics for solutions generated by the algorithms using CMSC data

ALGO	WRITE-IN	APL	NSFL	NSU	NSO	NSZL
IA	1504	75.03	985	217	0	302
IAMBM	1504	74.98	986	213	0	305
MAS	1504	80.06	993	335	0	176

The table shows that the MAS system wherein the scheduler agent did not generate any initial solution has the highest APL of 80.06.

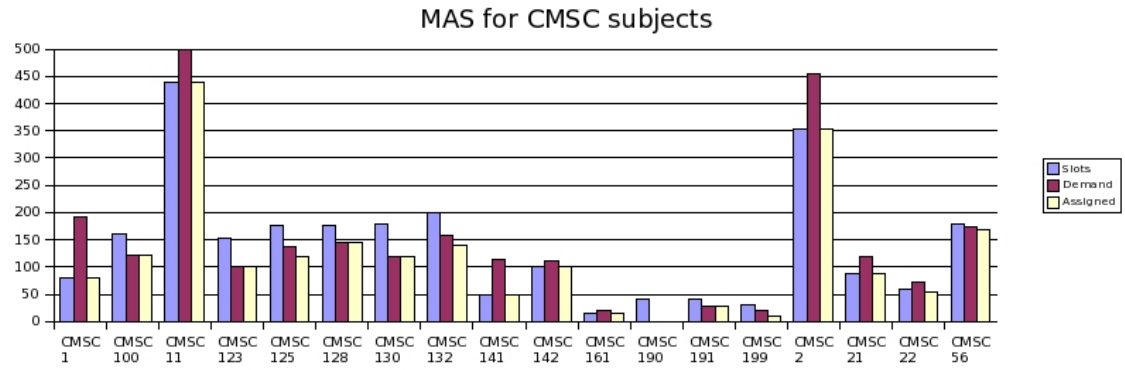

Figure 14. Available slots, demand, and assigned slots for CMSC subjects using MAS

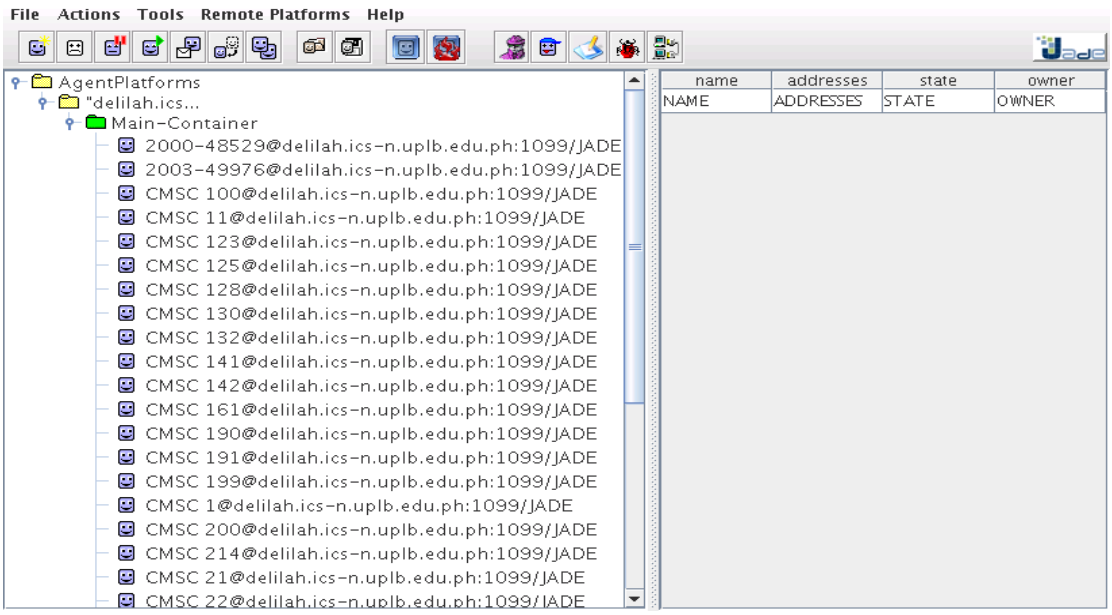

Figure 15. JADE Platform showing student agents and enlister agents

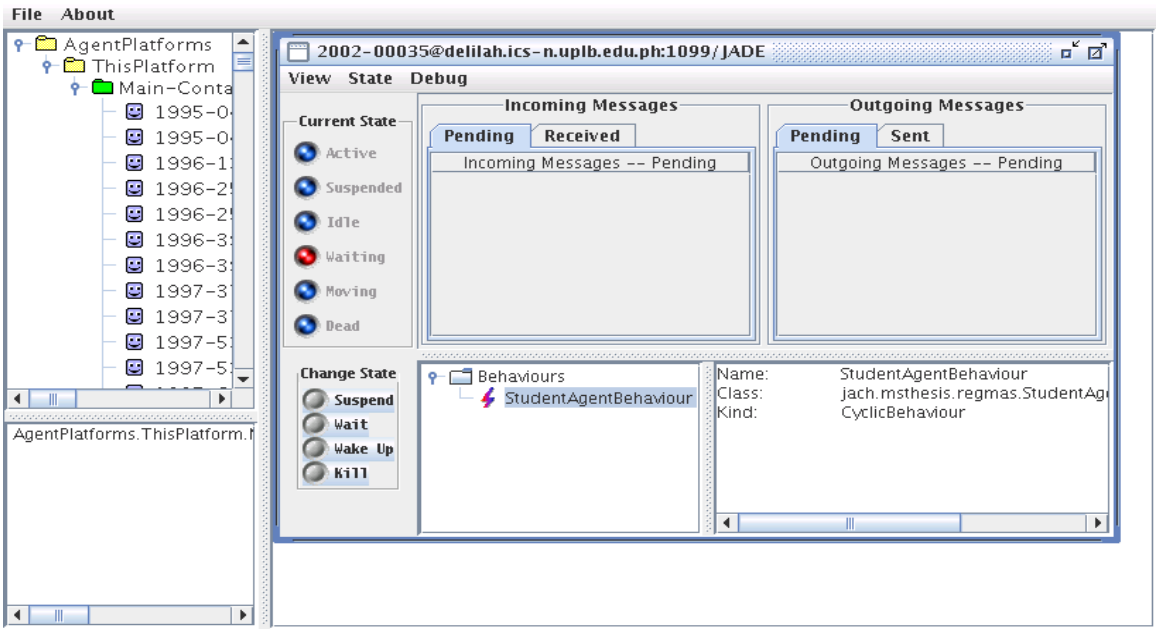

Figure 16. JADE Instrospector Agent showing the status of a student agent

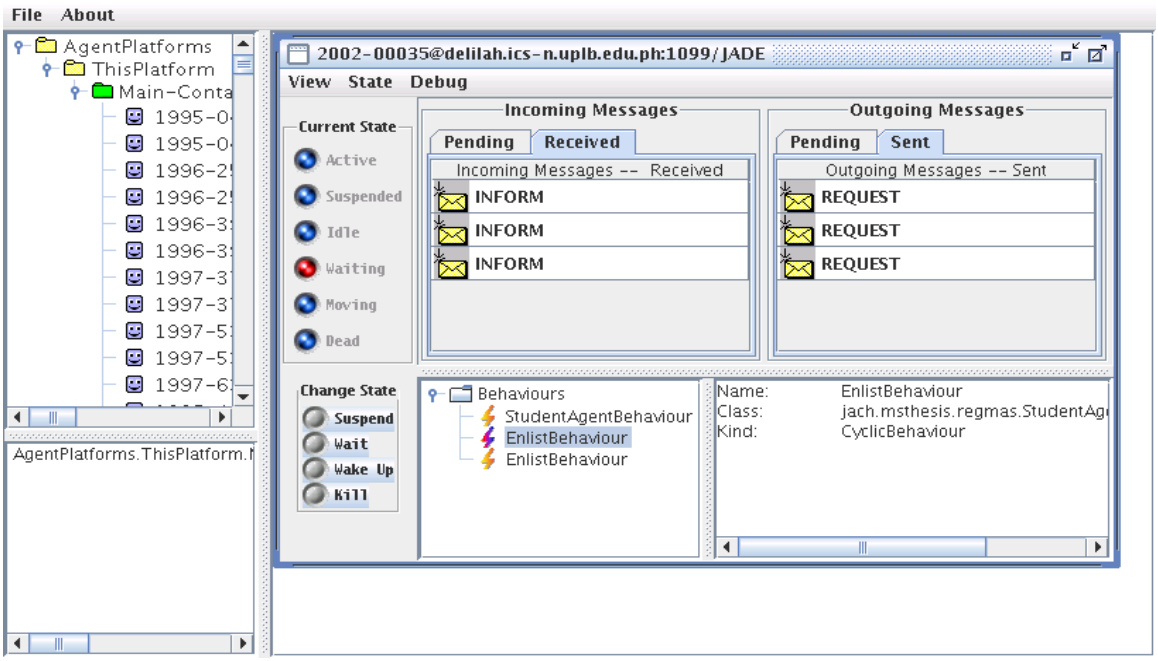

Figure 17. JADE Instrospector Agent showing communication messages, agent is currently enlisting

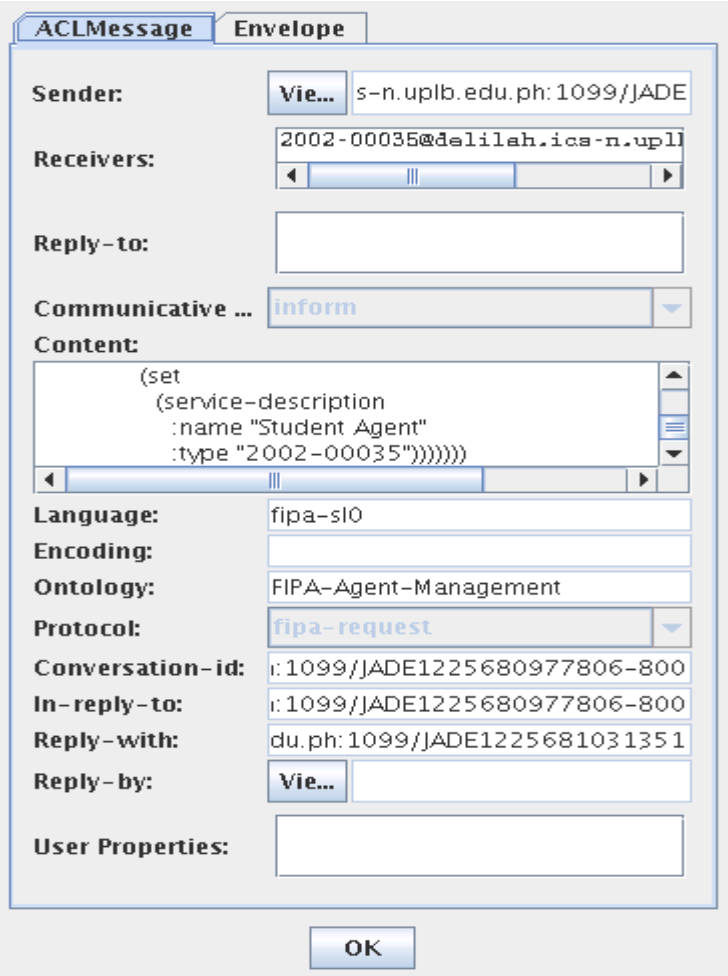

Figure 18. Details of a message being passed by a student agent.

SUMMARY AND CONCLUSION

This study has characterized the assignment of students to classes as the Student Sectioning Problem and has presented a mutiagent system framework for solving the problem. The multiagent system framework is composed of three types of agents namely a scheduler agent, enlister agents, and student agents. Central to solving the SSP using this framework is the scheduler agent which can generate an initial solution. In this study, an iterative algorithm is developed and augmented with a maximum bipartite matching solver to generate an initial solution. Student agents communicate with enlister agents to improve the initial solution by performing cancellation and enlistment operations. Using the proposed multiagent system framework allows for a distributed and concurrent problem solving process which is not present in the existing approaches based on local search and meta heuristics.

RECOMMENDATIONS

The proposed multiagent system framework for solving the Student Sectioning Problem is dependent on the behaviors exhibited by the agents, particularly the enlister and student agents. In this study, the student agent tries to obtain a complete assignment. After an iteration, if it has not obtained a complete assignment, it cancels all the slots it currently has but saves one slot in order to guarantee that it will at least have one assignment. This behavior can be modified for example instead of canceling all the slots it has, the student agent can define a preference ordering function on the current assignment in order to select the slots to cancel.

LITERATURE CITED

CARINO, R.L. 2007. Automation of Student Preregistration. Technical Report, Institute of Computer Science, University of the Philippines Los Banos.

CORMEN, T.H., C.E. LEISERSON, R.L. RIVEST, and C. STEIN. 2001. Introduction to Algorithms , 2nd ed. MIT Press: Massachusetts

DULDULAO, R.N. 2004. UPLB System One. http://systemone.uplb.edu.ph.

GAREY, M and D. JOHNSON. 1979. Computers and Intractability, A guide to the theory of NP-Completeness. W.H. Freeman and Company: New York.

GUINTO, D.T. 2004. University Timetabling using Tabu Search. M.Sc. Thesis, Institute of Computer Science, University of the Philippines Los Banos

MARAPAO, R.B. 2004. A Genetic Algorithm Approach for Solving the University Timetabling Problem. M.Sc. Thesis, Institute of Computer Science, University of the Philippines Los Banos.

MODI, P.J., W.M. SHEN, M. TAMBE and YOKOO, M. 2005. ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. Artificial Intelligence Journal. Vol. 161, No. 1-2,pp. 149-180.

RAO, A, M. GEORGEFF. 1995. BDI Agents from Theory to Practice. Technical Note 56, AAII. http://citeseer.ist.psu.edu/rao95bdi.html

RUSSEL, S.J., NORVIG, P. 2003. Artificial Intelligence: A Modern Approach, 2nd ed. Pearson Education, Inc: New Jersey.

WOOLDRIDGE, M., 2002. An Introduction to Multiagent Systems. John Wiley & Sons: Chichester, England.

YOKOO, M., E.H. DURFEE, T. ISHIDA and K. KUWABARA. 1998. The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. IEEE Trans. on Knowledge and Data Engineering, vol.10, No.5.

YOKOO, M. 2000. Algorithms for Distributed Constraint Satisfaction: A Review. Autonomous Agents and Multiagent Systems, Vol. 3, No. 2 ,pp. 198-212.