Designing a Multiagent System for Course-Offering Determination

Fuhua Lin[1], Wu Chen[2]
[1]School of Computing and Information System, Athabasca University
Alberta, Canada
oscarl@athabascau.ca

[2]Long View Systems, Alberta, Canada
william.chen@lvs1.com

Abstract. This paper describes the design of a multiagent system that facilitates course-offering decision making for a program in an institution. We model course-offering determination for upcoming semesters as a multi-winner election with exogenous constraints that is a problem of computational social choice in multiagent systems, which has rarely been considered. We propose a practical and effective approach to solving the problem, which is based on Contract-Net Protocol, Single Transferable Voting, and Monotonic Concession Protocol. We describe the goal model, agent behavior models, and the interaction protocols of the system through using the Gaia role model methodology, Tropos strategic actor diagram, Pseudo-code algorithms, and Agent Unified Modeling Language sequence diagram. The effectiveness of the approach and the implemented system has been showed with the initial experimental results.
Keywords: multiagent systems, agent-oriented software engineering, course-offering determination, and voting.

1 Introduction

Course offering determination (COD) can be defined as deciding on what courses of an academic program in an institution will be offering for the upcoming one or more semesters. There are many factors like the historic data in enrollments, budget and staffing constraints, which the university administration needs to take into account in order to provide a list of offering courses. A poorly designed course offering schedule could lead to a low enrollment to the program, delayed graduation, and unsatisfied students. Students in degree programs have various course selection preferences and priorities. A department of an academic institution may not be able to offer all courses in a program every semester, especially under contemporary fiscal and staffing constraints. Some courses could be only arranged every other semester or even less frequently. In the current course offering workflow, after a course delivery schedule for a new semester becomes available as the registration period is approaching, the student can select courses to be taken in the coming semester. The competing or even adversarial goals of students and the department as well as the mutability of those

goals indicate that COD is a complex constraint-satisfaction problem. Effective COD permits the efficient assignment of limited resources like faculty, labs, and classrooms while satisfying the desires of most students.

The multiagent system (MAS) approach [1] allows the representation of every principal in the system as a single autonomous agent with unique goals and permits decision-making based on the preferences of multiple agents [2]. The MAS approach is used to solve the COD problem because it has the following characteristics: (1) its optimal solution can change during calculation; (2) The relation between a user and the scheduling system lasts for a long period of time, which features a high degree of repetition, and may count for the possibility of learning by feedback; (3) The course scheduling is time consuming; (4) The process of scheduling for course-offering involves different parties, e.g. program administrators and students; (5). Program administrators consider job markets and the unpredictable nature of preferences students, which generate the need in course scheduling to adapt fast and flexibly to environmental variables and their changes.

Since the goals of the students and program administrators are different, therefore there are conflicts of interests between them, these conflicts should be resolved in a fair cooperative decision making manner. The main question of COD is:

> *What course offering determination strategy of a program in an institution maximizes the satisfaction of the students and the enrollment of the courses within budget?*

In this research, we model the COD problem as a *multi-agent constraint resource allocation problem* and design a mechanism to identify optimal solutions through using voting and agent negotiation. The system is expected to solve the problem in optimization and flexibility in a fair and rational way, balancing the competing needs of academic requirements, economics and student preferences, and considering the usability issues for all participants. This paper formally describes the design of the system for COD using Agent-Oriented Analysis and Design [3] and Agent-Oriented Software Engineering (AOSE)[4].

This paper is organized as follows. Following a brief survey of relevant published research, Section 3 will describe the goal model, scenarios, and interfaces between the system and its environment. This is followed by a session (Section 4) detailing the system architecture, particularly the various software agents and their roles and responsibilities. Section 5 describes with a detailed discussion of the voting and negotiating protocols used by the agents in solving the course-offering determination problem. Section 6 presents the implementation and initial experimental results. The paper ends with the conclusion and future work.

2 Literature Review

The majority of Artificial Intelligence applications to education address pedagogical tasks related to tutoring and personalized instruction, such as [5-6]. Significantly fewer publications are related to the decision-making or administrative tasks in education such as school choice [7], academic advising [8], and course timetable scheduling [9]. The MAS approach has proven an important and effective framework for intelligent educational systems, for example iHelp [10], program planning [11],

time table scheduling [12], and personalized study planning [13]. To the best knowledge of the authors, there is no significant extant work on solving the COD problem.

Voting procedures focus on the aggregation of individuals' preferences to produce collective decisions. There are many different voting rules; many of which are defined in the survey paper by Brams and Fishburn (2002) [14]. Lang (2007) considered voting and aggregation rules in the case where voters' preferences have a common preferential independence structure, and addressed the issue of decomposing a voting rule or an aggregation function following a linear order over variables [15]. Conitzer (2010) explained voting rules in more detail [16]. Plurality voting is one of the most common rules. In this scheme, the alternative with the greatest number of "first place votes" wins. Plurality voting does not require that voters provide rankings. This feature seems to be an advantage; however, this "elicitation advantage" means that it fails to account for relative voter preferences for any alternative other than its top choice. Other schemes produce winners that are more sensitive to relative preferences, among them are Positional scoring rules, Maxmin fairness, Copeland, Maximin, and Bucklin [17]. An obstacle to the widespread use of voting schemes that require full rankings is the informational and cognitive burden imposed on voters, and concomitant ballot complexity.

In this paper, we first model COD decision settings, e.g. modeling students as a group of self-interested agents, expressing their preferences [18] and then use a group decision-making protocol --- voting theory [19] to aggregate the preferences of the different participants toward a single joint decision. Single Transferable Vote (STV) is a staged procedure [19]. In practice, voters need not be required to rank all candidates. It has been shown that STV, in comparison with other voting schemes in actual use, is computationally resistant to manipulation [20]. The use of multiple fractional votes also provides another advantage – it makes the systems sufficiently computationally complex that it is resistant to manipulation [20] in the way that whole vote plurality systems might be [21].

Finally, we show how the negotiation techniques [22] is used to generate a set of course offerings for the next term that reflects those preferences, academic requirements, and economic necessities to mutual benefit between the department and the students.

COD is a multiagent resource allocation issue [23]. Thus it is needed to devise a protocol for it. There are two approaches for it: centralized such as combinatorial auctions [24] and distributed, e.g. Contract-Net Protocol (CNP) [25]. We can not directly implement COD system using auction-based resource allocation mechanism as the winner determination mechanism between auction based resource allocation and what we expect in the COD system is different. Auction based resource allocation is seeking for the highest price bid to win the resource. The winner is a buyer with the highest price bidding. In the COD system, however, the winner is a course (a resource from the seller) that has the most preferred enrolment from the students (buyers). CNP is mainly for solving the problem of distributing tasks to appropriate agents to execute the tasks for efficiency in general. It includes a negotiation protocol for interaction between agents like immediate response, direct contracts, request and information message, and node available messages. It can be used to form a team through mutual selection, exchange information in a structured way to converge on assignments. In

this research, we use it as a viable and flexible coordination mechanism for planning and enacting the collective course-offering decision making tasks.

3 Requirement Analysis

COD is concerned with the optimal assignment of the courses for a specific academic semester by program administration to meet the needs of students within budget and other resource constraints. Students in degree programs frequently have to balance personal career objectives, preferences, and financial and temporal constraints against degree requirements, course availability, and course inter-relationships. These and other constraints make the COD problem a complex one. The problem is made more complex by the fact that the constraints within which course offering determination is performed are fluid: students change their goals or are unsuccessful in completing prerequisite courses; faculty go on leave or modify research priorities; funding is awarded or withdrawn; other programs or institutions reserve or release facilities for instructional use. The fundamental business need of the research is for a program to identify those courses with sufficient interest to support the economics of offering them. To offer an optimal course list, information on learner needs, constraints, and preferences must be considered. To facilitate collaboration among learners, it can be desirable to allow students to share personal preferences and goals as well as create various learning communities/social networks during program study.

3.1 Goal Model

Students and program administrators have different goals. From the perspective of the student, course selection and planning is important in current educational environments. Selection of the 'right' course(s) can be a high-risk decision-making process because the cumulative effect of a series of choices made on successive semesters or quarters may impact their college major selection, their ability to take additional course work, their graduation date or even their career direction and future employment opportunities. From the perspective of educational providers, effective course offering determination permits the efficient assignment of limited resources like faculty, laboratories, and classrooms while at the same time satisfying the needs and desires of most students. Consequently, there is an inherent tension between the needs of the student body and the available resources of the educational provider. Fig. 1 illustrates the goal model of the system proposed consisting of Students and Program Administrator. The goal of a program student is to "complete a program" while the goal of a program administrator is to "determine a course-offering schedule". The goal "complete a program" is associated with a quality goal "study desirable courses and complete a program efficiently". The goal "determine course offering schedule" is associated with "offer courses cost-effectively." The goal "complete a program" has the subgoals "select preferred courses" and "express preferences." The goal "determine course offering schedule" has the subgoals "aggregate students preferences", "calculate cost and revenue", and "decide courses to be offered".

The competing goals between students and resource providers mean that COD is a complex dynamic constraint-optimization problem where the optimal solution can change during calculation. Consequently, the use of a multi-agent system that continually re-evaluates the current constraint state as reported by constituent agents in order to maintain an appropriate solution plan as the environment changes.

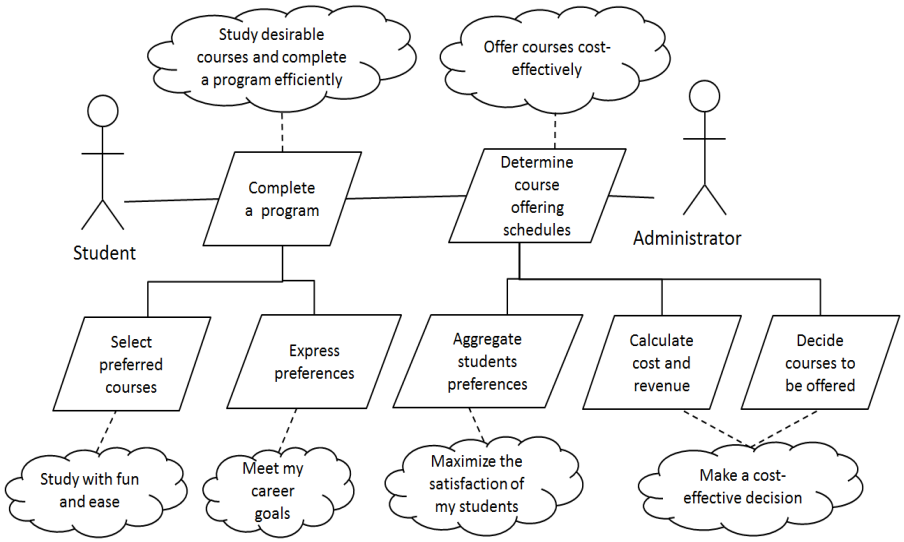

Fig. 1. The goal model for a course-offering decision support system.

3.2 Agents

We designed a MAS system that consists of an administrator (AD) agent, a group of student (SA) agents, and a student representative (SR) agent. This correlates with an academic program, where a course scheduling process is initiated by having the program administrator determine the priority of courses available in the program, based on expressed student needs, preferences, and goals. The agents have distinct areas of concern and intent, but collectively interact to generate a set of recommendations for courses to be offered that will be satisfactory to most students while fitting within the operational limitations of the offering program. A SA agent has three major responsibilities: (1) representing the student principal's interest in interactions with other agents; (2) generating plans for their principals; and (3) generating course selection requests for their principals. A SR agent is instantiated for any identifiable student group with shared interests and resources. Typically, this would be one per academic program. The SR agent has the following major responsibilities: (1) Managing voting among student agents, ensuring fairness. (2) Representing the student body to other agents, particularly the AD agent. The AD agent is the representative of the program administrations calculating the needs of the academic department as determined by factors such as course delivery policies,

budget, and resource availability. It is responsible to: (1) Provide executive control and oversight for the system; (2) Enforce resource and other course availability constraints; (3) Inform other agents of those constraints; (4) Negotiate with the SR agent to provide an optimal set of course offering recommendations to the offering academic program.

3.3 Scenarios

Within this system, for each semester, before the course registration starts, the program administrator of the department delegates the task of initial selection to his/her AD agent, which performs this task using course dependency graphs, past offerings, and departmental obligations.

The AD agent collects requirement information from the program administrator, which determines a set of required courses for the next term as well as the proposed budget for course delivery in the next term. At the same time, each SA agent generates a study plan based on the program study preferences of the student, identifies "*ready-to-take"* courses of the student, and captures course selection preferences of the student. Once all SA agents complete the actions mentioned above, they send their votes to the SR agent. And then the SR agent aggregates the votes and generates the set of all ranked preference ordering over courses as a group decision.

Once the AD agent has this information, it initiates a one-to-one agent negotiation with the SR agent. The negotiation then proceeds iteratively between the two agents, the AD agent attempting to maximize the course enrolments and minimize delivery overhead, and to maximize staff efficiency of the offering, and the SR agent attempting to maximize the availability of courses desired or needed by students to complete their programs.

4 Architectural Design

In the current model, there are three roles coming into play in COD system –SA agents, SR agent, and AD agent. As shown in the Table 1 below, the main activities of SA Agent is to present students' interest, generate a study plan, and select course they want for vote. They read the interests from the students and select courses based on the student's interest and offering course list. The sequence of activities of SA agent is the more than one time of repeat on PresentInterest, GeneratePlan, CourseSelection, GenerateVotes, and CalculateDisutilitySA. The constraint on these actions is the number of selected course must be less than or equal the total number of interest presented courses.

Please note that unlike obligations in normative framework refer to the consequence of some action, responsibilities in Gaia methodology [3] we use in this paper is one of attributes of a role and they determine functionality of an agent. Responsibilities include liveness properties and safety properties. In order to realize responsibilities, a role has a set of permissions. In Gaia, resources are thought of as relating to the information or knowledge the agent has. That is, in order to carry out a role, an agent will typically be able to 'access', 'modify' or 'generate' certain information. This specification defines two types of permissions for SA agent: read,

change, and generate. It says that the agent carrying out the role has permission to *access* the value Presentinterests, and has permission to both *change* (*read and modify*) the value CourseSelection. Also, the role is the producer of Plan, Votes, and Disutility.

Table 1: SA Agent role model.

Role Schema	SA Agent
Description	SA Agent represents the student principal's interest, generates plans, and requests course selection for their principals.
Protocols & Activities	PresentInterest, GeneratePlan, CourseSelection CalculateDisutilitySA(), GenerateVotes()

Permissions	Reads Presentinterests // all interested courses presented
	GeneratePlan // retrieve info from the student study plan
	Changes CourseSelection //select courses from the list
	GenerateVotes() // generate vote count on the course
	CalculateDisutilitySA() // Calculate the SA agent Disutility

Responsibilities	Liveness: SA Agent = (PresentInterest.GeneratePlan.CourseSelection. Safety: GenerateVotes. CalculateDisutilitySA)+ Number_of_SelectedCourse <= Number_of_InterestPresente.

The role schemas of AD agent and SR agent are similar to the SA agent role model.

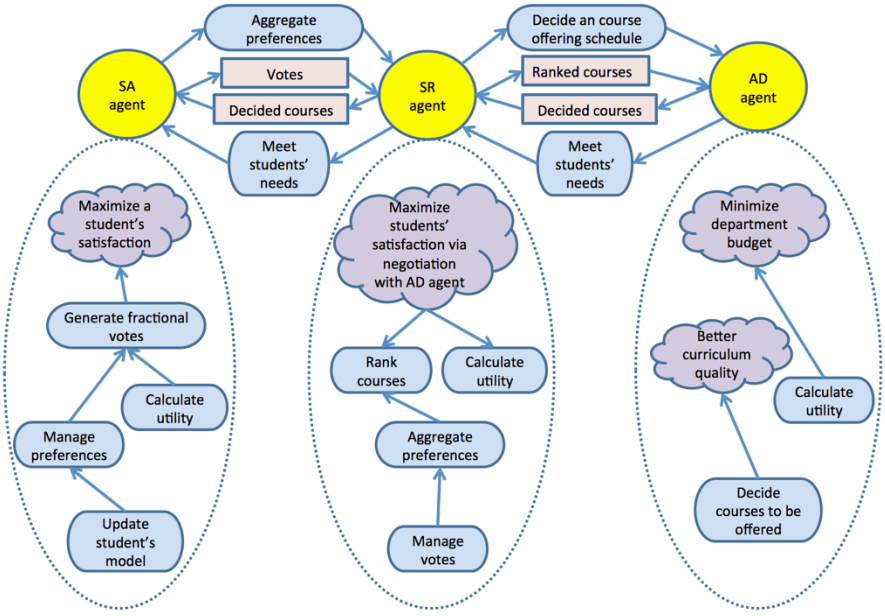

Fig. 2. A Tropos goal diagram for SA agent, SR agent, & AD agent.

Three actors are identified: SA agent, SR agent, and AD agent. Fig. 2 illustrates a Tropos (http://www.troposproject.org/) goal diagram for three types of agents in the

system proposed. Two dependencies are modeled as goal dependencies: Aggregate preferences and Decide an course offering schedule. In a goal dependency, one actor depends on another actor for achieving a goal. Four dependencies are modeled as resource dependencies: vote, decided courses, ranked courses, and decided courses. The goals of the three types of agents are decomposed into some subgoals. Soft goals like Maximize a student's satisfaction, Better curriculum quality are considered, which may contribute to the achievement of a goal or prevent its achievement. The soft goals are represented as clouds in the figure.

5 Detailed Design

In order to design and implement a system that provides support for COD, there are a number of critical pieces that must be assembled into an integrated whole: (1) An effective way of collecting and representing student preferences with regard to their current and future studies as well as their plans following graduation; (2) A means of modeling relationships and interdependencies amongst courses, programs of study and learning objectives; (3) An efficient and fair means for balancing the competing desires and needs of students; and (4) A fair means of apportioning the available faculty, facility and funding resources of the program, faculty or school providing education.

5.1 Preference Models of Student Agent

Our initial investigation was through interviews and anecdotal discussions with current students and staff members. We had strong indications that the desirability of a particular course was not independent of other course's availability. It appeared that course selection preferences are more properly thought of as conditional preferences [18]. Furthermore, there were a variety of ways and degrees of complexity that individual students might express the way in which they determine which courses they want to take in the upcoming term. We identified three distinct models of how students described their preferences with regard to course offerings, which we have termed: *precedence*, *grouping*, and *progression* [25].

Precedence. The first model, precedence, simply referenced a most-preferred course, a next-most preferred, and etc. However, after a comparatively short list of courses, the students lapse into a don't-care state along the lines of "if none of those are offered, then it doesn't matter". This sort of preference model is common in cases where the student only plans on taking one or a very few courses.

A SA agent generates precedence by using the degrees of desirability of the courses in his/her study plan prior to the voting process [8]. To do this, the SA agent determines the list of all courses in the program that that student can legally and preferably take, C_{ready}, which consist of all incomplete courses of the student for which all pre-requisites have been completed.

The SA agent receives a list of courses from the AD agent that will not be offered, $C_{not\_to\_offer}$. The SA agent receives a list of courses that will already be recommended from the AD agent, $C_{recommended}$. These are similarly removed from the "legal" course set –but are treated as though they will be offered for purposes of other decisions.

Thus, the SA agent automatically generates $C_{precedence}$ by removing $C_{not\_to\_offer}$ and $C_{recommened}$: $C_{precedence} = C_{ready} - C_{not\_to\_offer} - C_{recommended}$ and then the elements in $C_{precedence}$ are ranked by the degrees of *desirability* and shown in the *precedence* interface so that the student can modify them if wish. It is represented as

$$C_{prededence} = \{(C_i, w_i): w_1 \geq w_2 \geq ... \geq w_n, c_i \in C\}.$$

here, $c_1 \succ c_2 \succ ... \succ c_n$. Here C is the set of all courses in the system. Integer n is the number of all elements in $C_{precedence}$.

Grouping. This model was more common amongst students planning on taking several courses. With it, students express their desires in terms of sets - a group of courses is desired en masse, and if not all courses are available, and then the remainder are less desirable. *Grouping* = $\{G_i \subset C, 1 \leq i \leq l\}(l \geq 0)$. We use an array to represent the grouping preference: $A_{grouping} = \{ (g_{ij})_{nxn}$: $g_{ij} = 1$, if c_i and c_j are in the same group, otherwise 0; $g_{ii} = 0\}$.

Progressions. Finally, there is the case of those students that plan their program *progressions* sequentially – their desire is to complete some set of courses, then progress to another set, etc, which we have termed the progression model. This sort of preference closely approximates the way in which academic departments model academic progression and is common in the case where students are in a full-time degree program or wish to systematically complete a program. This progression model is also most similar to that expressed in the CP-net [18].

We use a directed acyclic graph (DAG), $P = (C, A)$, to represent the progression model. Set C represents all courses in the program. Set A consists of arcs. An arc $e = (c_1, c_2)$ is considered to be directed from c_1 *to* c_2; c_2 is called the *head* and c_1 is called the *tail* of the arc; c_2 is said to be a *direct successor* of c_1, and c_1 is said to be a *direct predecessor* of c_2. If a path made up of one or more successive arcs leads from c_1 to c_2, then c_2 is said to be a *successor* of c_1, and c_1 is said to be a *predecessor* of c_2.

For implementation, we use an array to represent P:

$$A_{progression} = (p_{ij})_{n \times n}. \quad p_{ij} = 1, \text{ if } (c_i, c_j) \in A, \text{ otherwise } p_{ij} = 0.$$

Let $\vec{T}_{taken} = (t_1, t_2, ..., t_n)$ represent the status of the courses about whether or not the student has completed. $t_i = 0$ means that c_i has not been taken while $t_i = 1$means that c_i has been taken and completed by the student. Then$\vec{T}_{taken} \in A_{progression}$ represents the readiness of the courses in terms of the progression preference of the student. That is, if $r_j \triangleq \sum_{i=1}^{n} t_i \times p_{ij} \geq 1$ then c_j is ready to take according to student's progression preference.

5.2 Generating Fractional Votes by Student Agents

Based on the preferences expressed by the principal on one or more of our preference models, the SA agent automatically generates a set of fractional votes for each round of an election. From the precedence set, the highest *ranked* course that can be preferably and legally taken is added. From the *grouping* interface, all courses from groups in which all member courses can legally be taken are added to the list.

From the *progression* interface, all legal courses that are either in the first block of a progression or whose predecessor block contains only courses that have already been taken are added to the list. Each of the courses on the list then gets a relative fraction of the agent's single vote for that round, with repeat courses getting a

proportionately higher portion of the vote. The vote is recalculated for each round, and can be affected by the results of previous votes.

For the sake of illustration, let's consider a real case. Suppose $C = \{c_i\}_1^{10}$. Let us assume that a student has previously completed c_1 and c_2. According to curriculum checking, could legally take c_3, c_4, c_5, not including c_7. For the coming semester, the student expressed the following preference: *Precedence:* $\{(c_3, .88), (c_4, .56), (c_5, .55)\}$; *Grouping:* $\{c_3, c_4\}$, $\{c_5, c_7, c_9\}$; *Progression:* $\{c_1\} \rightarrow \{c_2\} \rightarrow \{c_3, c_7\}$. From the precedence expressed, c_3 would be added with a weight 0.88, c_4 with a weight 0.56, c_5 with a weight 0.55 to that student's vote list. From the groups identified, c_3 and c_4 would be added to the vote list, as the first group can be fulfilled. However, no entries from the second group would be added, as c_7 is not legal (in this case, the status of c_9 is irrelevant). Finally, from the progression, c_3 would be added to the vote list, however c_7 would not be added until it is legal later. Each of the courses on the list then gets a *relative* fraction of the agent's single vote for that round, with repeat courses getting a proportionately higher portion of the vote. The total vote set for our fictitious student is thus: $\{(c_3, 1), (c_3, 1), (c_4, 1), (c_3, .88), (c_4, .56), (c_5, .55)\}$. *Let* v_i represent the fractional vote on course c_i. The student's vote is then rationalized into the votes as follows: $v_3 = (1+1+0.88)/(1+1+1+0.88+0.56+0.55) = 0.58$; $c_4 = 0.31$, $c_5 = 0.11$. The algorithm for generating fractional votes is listed as follows:

```
ALGORITHM: Generate Fractional Votes
input:  C, C_precedence , A_grouping  A_progression
T⃗_taken= (t_1, t_2, ..., t_n), course completion status vector
R⃗= (r_1, r_2, ..., r_n), course progression readiness vector
Returns: V = {v_i: 1 ≤ i ≤ n}
begin
generate C_precedence;
retrieve A_grouping and A_progression
  for  i = 1 to n do // consider "Precedence Model"
       if c_i in C_precedence then  v_i ← w_i ; end if
  end for
  for i = 1 to n do  // consider "Grouping model"
       if not(c_i in C_precedence) then
         for j = 1 to n do  g_ij ← 0; g_ji ← 0; end for
  for i = 1 to n do
        for j = 1 to n do
          if g_ij = 1 then v_i ← v_i +1; end if
     end for   end for
    for  j = 1 to n do// consider "Progression Model"
        for i = 1 to n do  r_j ← t_i·p_ij + r_j; end for
        if (t_j = 0) and (t_j = 1) then v_j ← v_j + 1 end if;
end for
end
```

5.3 Interaction Protocols

We use an Agent Unified Modeling Language (AUML) (www.auml.org) sequence diagram (see Fig. 3) to depict the interaction protocols for coordination, voting, and agent negotiation.

Determining Negotiable Courses: First, AD agent starts the process by sending a message to the administrator once the registration period is approaching to acquire the must-to-offer list C_0, the must-no-to-offer list C_{-1}, and maximum number of courses that can be offered n_0. The administrator determines them through using the past experience, the course dependency, and the department obligations.

At the same time, the students then will develop their study plans according to the initial course lists, study preferences, ready-to-take courses, and course selection preferences.

The AD agent determines a list of negotiable courses C'' as the first proposal. And then AD agent will send $C'' = C_0/C_{-1}$ to all SA agents. And then the student agents will notify all students to register courses from the initial course offering list $C_f \leftarrow C''$.

The AD agent computes the initial disutility U_{AD} based on the registration. It means the difference between the actual operating cost *Cost* and the targeted budget $Cost_{ideal}$, and determine a list of negotiable courses $C' = C/(C_0 \sqcup C_{-1})$.

Voting: SR agent uses CNP to work with all SA agents to coordinate the voting and election process. The study plan of each student is passed to his/her SA agent and is used to generate precedence relation and be combined with other preferences to be turned into a fractional vote to their SR agent. SR agent aggregates the fractional votes from all SA agents and generates a set of all ranked preference ordering over the course set C', denoted as *STV-k*, through using an algorithm based on STV [19] and the number of the slots to fill, i.e. $k = n_0 - |C'|$. And then it sends the to AD agent. This allows the protocol to converge more rapidly on an acceptable solution by initially jumping by several courses before dropping down ultimately to one course at a time for the final negotiations. It also allows us to use STV more optimally, as its value is more readily apparent in elections where several candidates will be elected. AD agent sends STV-k to the administrator.

Based on a monotonic concession protocol (MCP) [22], a negotiation protocol between AD agent and SR agent to determine an optimal offering course list is developed.

For each round of agent negotiation r, AD agent proposes δA (the first round its value is $C'' \sqcup \Phi$) with its utility $U_{AD}(\delta A, r)$ while SR agent proposes $\delta S = C'' \sqcup$ *STV-k*' and computes disutility $U_{SR}(\delta A, r)$ *and* $U_{SR}(\delta S, r)$. The algorithm for computing U_{SR} is listed in Appendix A.

SR agent determines if $U_{SR}(\delta A, r) < U_{SR}(\delta S, r)$, which means that for SR agent offering more courses will not decrease the dissatisfaction of the students and thus does not need to add more courses. The negotiation ends. Otherwise, AD concedes and selects the first course in *STV-k*, *c*, and adds *c* to the new proposal δA, the course offering C_f list is updated. And then the student agents will notify the students to register courses from the current course offering δA. It is worthy to mention that for AD agent, due to the fact that we can not compute $U_{AD}(\delta S, r)$ without knowing the registration, we can not check if $U_{AD}(\delta A, r) > U_{AD}(\delta S, r)$ to apply standard MCP. Our approach is that if there is a need to increase some courses, we select the first course in *STV-k* and add it to new proposal δA and get the enrollments for the courses.

The AD agent computes the initial utility $U_{AD}(\delta A, r)$ based on the registration and the budget, and updates ranked negotiable courses *STV-k* with *STV-k* /{*c*}. The negotiation goes to the next round. For each round of the agent negotiation, AD agent needs to check if *STK-k* is empty, or $U_{AD}(\delta A, r)$ still meets a satisfied predicate AD_{sat}. $AD_{sat} = (U_{AD} <= \Delta U_{AD})$. ΔU_{AD} is a pre-set threshold. If AD_{sat} is true, the AD agent is satisfied with the current offering. Also, SR agent checks a satisfied predicate, SR_{sat} which indicates whether the aggregate disutility is less than the permitted variance $SR_{sat} = (U_{SR} <= \Delta U_{SR})$. ΔU_{SR} is a pre-set threshold. If SR_{sat} is true, the SR agent is satisfied with the current offering.

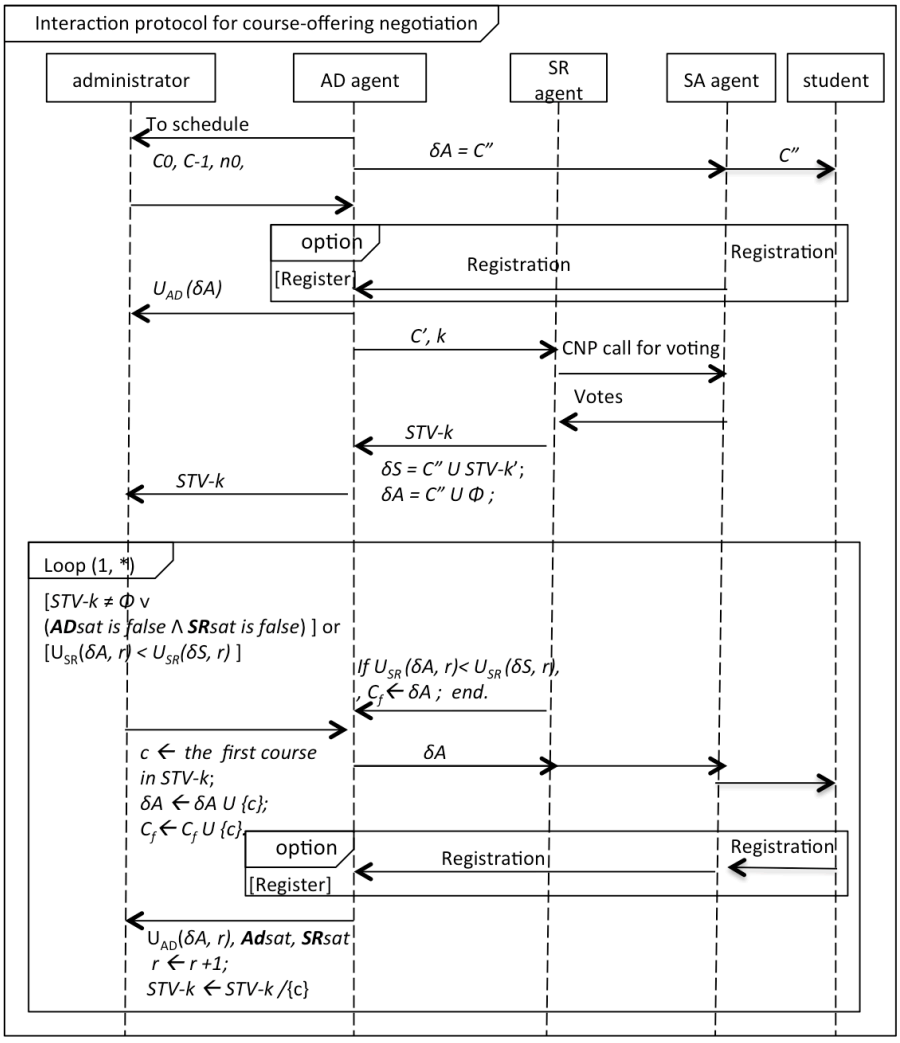

Fig. 3. AUML sequence diagram for interaction protocol of the negotiation.

If A_{sat} is true and the SR agent has indicated that the last round satisfied it, the negotiation is concluded and the AD agent records the final offering list and sends it to the SA agents. If the A_{sat} is false, the negotiation will always continue. If the AD agent is satisfied, but the SR agent is unsatisfied, the negotiation will continue until the count of courses in a proposal reaches zero. In this way, the negotiation converges fairly rapidly to a set that is near the budgeted amount for the department, while trying to add courses most likely to fit into the plans of the majority of students.

6 Implementation

To evaluate the proposed approach, a prototype was implemented with JADE (http://jade.tilab.com) and Jason (http://jason.sourceforge.net/wp/), which allows simulating different scenarios by varying several parameters, such as the course number of the program, the divergence of preferences for course selection, or the must-offer courses due to emergence cases. We simulated a collection of master program students consisting of from 30 to 90 MSc in Information Systems students of Athabasca University in Canada. Fig. 4 shows a screen shot of entering student course selection preferences. Fig. 5 is an example of voting result generated from one of the experiments.

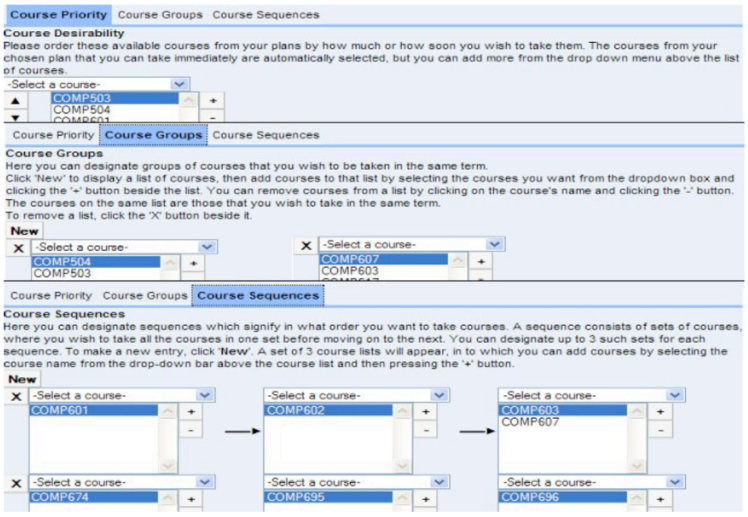

Fig. 4. a screen shot of student course selection preferences.

Given that a course set $C = \{c_i\}_{i=1}^{n_c}$ to be offered in a semester in the program, the actual cost to be paid for the course offering is calculated as follows:

$U_{AD} = |Cost(C) - Cost_{ideal}|$, $Cost(C) = b(n_c) + r\sum_{i=1}^{n_c} h_i$, where b is the base salary for one course to be paid to the instructor for the course, e.g., b = \$5000; r is the amount of money to pay to the instructor for one student minus the course fee, e.g. r = \$500; h_i is

the number of the students who will take course c_i for $1 \leq i \leq n_c$, and C_{ideal} =$75,000.; n_c = 30, 50, 90; $C_0 = \{c_{501}, c_{503}, c_{504}, c_{601}, c_{695}\}$; $C_{-1} = \{c_{602}, c_{604}, c_{617}, c_{636}\ \ c_{637}, c_{682}\}$; ΔU_{AD} = $100; $\Delta U_{SR} = 3$.

In Round 1: $C_1 = C_0 \cup \{c_{602}, c_{605}, c_{607}, c_{610}, c_{648}, c_{660}, c_{667}, c_{689}, c_{691}\}$. Getting h_i, $U_{AD}(C_1)$ = $11500.0. In Round 2: c_{691} is chosen in this round, with a voting result of 60. Getting h_{691}= 64 since it is the first course chosen, and many students have it as their first pick due to it being on many plans and not having many prerequisites. $U_{AD}(C_1)$= $15000.0. Comparing C_{ideal}, the Administrator is still not satisfied with required courses. The SR agent is also unsatisfied with course offering as SR Weight: 10.31. After Round 3 and Round 4, in Round 5, the required 9 courses are: c_{691}, c_{605}, c_{648}, and c_{667}. c_{689} is chosen in this round, with a result of 2.0. U_{AD} = $48,000. So the AD agent is still unsatisfied with the list. The SR agent is now satisfied with course offering. The SR agent's weight is 0.86. Negotiation concluded with a list: $C_1 = C_0 \cup \{c_{691}, c_{605}, c_{648}, c_{667}\}$. The size limit of a class is not considered.

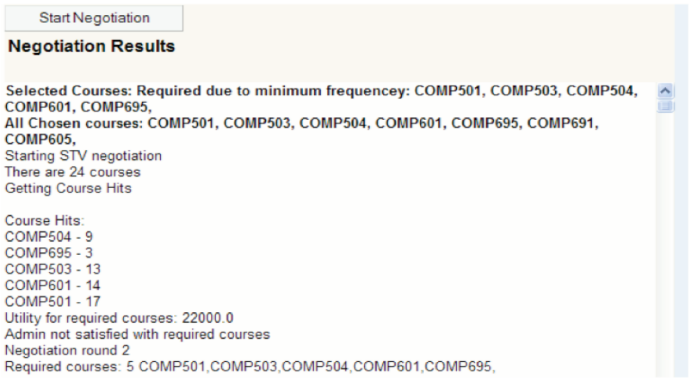

Fig. 5. An example of voting result generated.

7 Conclusions

We have presented the design of a multiagent system for course-offering determination. It includes models of agent goals and behaviors, and protocols of agent-based coordination in dynamic decision making of individual students and the group decision-making of program administrators. One of the contributions of this paper is the novelty of the problem domain of determining a group of course offerings in educational institutions. The work can stimulate discussion of alternative points of view for how to solve the problem and lead to further discoveries. The second contribution of this research is the multi-agent system architecture proposed and the algorithm for the reasoning capability of the student agent, preference elicitation and inference algorithm. Finally, there is significant value to the mechanism design — the administrative agent's coordination capability to recommend suitable course offerings to departments in academic terms; in which multiple student preferences are aggregated using STV for the aggregation of multiple student preferences, the course budget of the department, and the derived cost of the courses offered. Each student's

preferences are translated into fractional votes that inform a negotiation process bounded by academic and resource constraints. The future work includes more testing and deploying the system to turn it into a practical application. We will study multi-winner election problem with exogenous constraints in other application domains.

Acknowledgement

We thank anonymous reviewers for their constructive feedback, NSERC for the financial support to the research, and AJ Armstrong and Alex Newcomb for their help.

References

1. Weiss, G., (Ed.) Multiagent Systems, A modern approach to distributed artificial intelligence. MIT Press. (ISBN 0-262-23203-0) (1999)
2. Conitzer, V., Making decisions based on the preferences of multiple agents. *Comm. ACM*, 53(3), 84-94, (2010)
3. Wooldridge, M., Jennings, N. R., & Kinny, D., The Gaia Methodology for Agent-Oriented Analysis and Design, Journal of Autonomous Agents & Multi-Agent Systems, 3, 285-312, (2000)
4. Winikoff, M. & Padgham, L., Agent-Oriented Software Engineering, chapter 15, in Book: Multiagent Systems (edited by G. Weiss) 2nd edition, MIT Press, 2013.
5. Graesser, A., Chipman, P., Haynes, B., & Olney, A. AutoTutor: an intelligent tutoring system with mixed-initiative dialogue. IEEE Trans. on Education, 48(4), 612–618 (2005).
6. Mitrovic, A., & Ohlsson, S., Evaluation of a constraint-based tutor for a database language. International Journal on Artificial Intelligence. 10, 238-256 (1999).
7. Wilson, D. C., Leland, S., Godwin, K., Baxter, A., Levy, A., Smart, J., Najjar, N., and Andaparambil, J., SmartChoice: An Online Recommender System to Support Low-Income Families in Public School Choice. AI Magazine, 30(2), 46-58 (2009).
8. Lin, F., Leung, S., Wen, D., Zhang, F., Kinshuk, & McGreal, R., e-Advisor: A Multi-agent System for Academic Advising, Workshop on Agent-Based Systems for Human Learning and Entertainment (ABSHLE) at Autonomous Agents and Multi-Agent Systems (AAMAS) 2007, Honolulu, Hawaii, USA
9. Oprea, M., MAS UP-UCT: A multi-agent system for university course timetable scheduling. Inter. J. of Computers, Communications & Control, II(1), 1024–1020 (2007).
10. Vassileva, J., McCalla, G. , & Greer, J. , Multi-Agent Multi-User Modeling In I-Help, *User Model. User-Adapt. Interact.* 179–210 (2003)
11. Hamdi, M. S., MASACAD: A Multiagent-Based Approach to Information Customization. IEEE Intelligent Systems, 21(1), 60-67 (2006).
12. Tariq M., Mirza M., Akbar R., Multi-agent Based University Time Table Scheduling System (MUTSS). Inter. J. of Multidisciplinary Sci. and Engg., 1(1). 33-39 (2010)
13. Vainio, A., & Salmenjoki, K. Improving Study Planning with an Agent-based System. Informatica, 29, 453–459 (2005)
14. Brams, S. J. and Fishburn, P. C., Voting Procedures. In K.J. Arrow et al. (eds.), Handbook of Social Choice and Welfare, Elsevier, 173-236 (2002)
15. Lang, J., Vote and aggregation in combinatorial domains with structured preferences, in Proc. of the 20th Inter. Joint Conf. on AAAI, pp. 1366-1371 (2007).
16. Conitzer, V., Making decisions based on the preferences of multiple agents, Comm. ACM, vol. 53, no. 3, pp. 84-94, (2010)
17. Lu, T. and Boutilier, C. Robust Approximation and Incremental Elicitation in Voting Protocols, IJCAI'11, Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence – Vol. 1, 287-293 (2011)

18. Boutilier, C., Brafman, R. I. , Domshlak, C., Hoos, H. H. , and Poole, D. , CP-nets: A tool for representing and reasoning with conditional *ceteris paribus* preference statements, J. Artificial Intelligence Research (JAIR), 21, 135–191 (2004)
19. Bartholdi, J. J. and Orlin, J. B, Single Transferable Vote Resists Strategic Voting. Social Choice and Welfare, 8, 341-354 (1991)
20. Walsh, T. An Empirical Study of the Manipulability of Single Transferable Voting, Proceedings of the 2010 conference on ECAI. (2010), 257-262
21. Sandholm, T. Vote elicitation: Complexity and strategy-proofness. In AAAI-02 proceedings, 392–397 (2002)
22. Rosenschein, J. S. & Zlotkin, G. (1994). Rules of Encounter: Designing Conventions for Automated Negotiation among Computers, MIT Press.
23. Lin, F., Armstrong, A. J., & Newcomb, A., A MAS Approach to Course Offering Determination, Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conferences on, 3, 331-336 (2012)
24. Cramton, P., Shoham, Y., & Steinberg, R., eds., Combinatorial Auctions. MIT Press, (2006).
25. Smith, R. G., The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Trans. on Computers, C-29(12), 1104-1113, (1980)

Appendix A

ALGORITHM CalculateDisutilitySA(C_{offer} ,U_{SA})
input: C_{offer}; //current *offering list*
$C_{student}$;//*course set that the student plans to take in the* upcoming term *in the precedence preference model.*
variables:
output: U_{SA}
$U_{SA} \leftarrow 0$;
for each course c in $C_{student}$
if $c \in C_{offer}$ **then**
$U_{SA} \leftarrow U_{SA} + 0$; *// The smaller is the value R, the happier is the student.*
else $U_{SA} \leftarrow U_{SA} + 1$; **end if**
return U_{SA};

ALGORITHM CalcuateDisutilitySR(l_1, l_2, U_{SA}**)** *// Calculate* U_{SR} *– Collective Disutility of SR agent.*
input: $\{U_{SAi}\}$
variables: $\bar{x}$ *: arithmetic mean; the average of* U_{SA}.
σ *: standard deviation of the reported* U_{SAi}. l_1, l_2 **:** weights for $\bar{x}$, and σ *used to tune the contribution of each component.* $l_1 + l_2 = 1$. Initially, they both are set to 0.5, which can be optimized through a machine-learning algorithm like the genetic algorithm.
for each SA_i **CalculateDisutilitySA**(C_{offer}, U_{SAi}) **end for**
$\bar{x} \leftarrow$ the average value of U_{SAi};
$\sigma \leftarrow$ *The deviation sigma* σ;
$U_{SR} \leftarrow l_1\bar{x} + l_2\sigma$;
return U_{SA} ;