A Multiagent System Framework for Solving the Student Sectioning Problem

[Extended Abstract]

Joseph Anthony C. Hermocilla
Institute of Computer Science
University of the Philippines Los Baños
College 4031, Laguna, Philippines
jachermocilla@uplb.edu.ph

Eliezer A. Albacea
Institute of Computer Science
University of the Philippines Los Baños
College 4031, Laguna, Philippines
eaalbacea@uplb.edu.ph

ABSTRACT
Student sectioning is the assignment of students to classes in such a way that no classes assigned to a student conflict in schedule and no class exceeds a specified class size. This paper proposes a multiagent system framework for solving the Student Sectioning Problem.

Keywords
student sectioning, multiagent systems, algorithms

1. INTRODUCTION
Student sectioning is the assignment of students to classes in such a way that no classes assigned to a student conflict in schedule and no class exceeds a specified class size. It is an important problem that a university must address when automating its student registration process, especially in universities with large number of enrollees and classes. The student sectioning problem is usually treated as a subproblem of the more general timetabling problem.

We define the Student Sectioning Problem as a tuple $SSP = (A, B, C, D)$ where A is a set of students with elements a, B is a set of subjects with elements b, C is a set of classes with elements a pair $c = (b, section)$, and D is a set of $write-in$ with elements $d = (a, b)$. We specify the attribute $timeslot$ to a class c. We define $slot$ as a pair $t = (c, n)$ and we let E be the set of all slots. The $classize(c)$ is the number of slots with c in the elements of E. We define an $assignment$ as a pair $f = (d, t)$ such that given a write-in d and the slot set E, $(b, section)$ is in C and c is in t. We also define a predicate $conflict(a, b)$ over a set of assignments Q such that given any two assignments f_1 and f_2 in Q, it returns $true$ if the timeslots of the c in f_1 and c in f_2 are the same and $false$ otherwise. The predicate $full(c)$ over a set of assignments Q returns $true$ if the number of assignments in Q which include c is greater than $classsize(c)$ .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 of student a. The union of all X_a for all a in A is the set S. The *classlist* for a class c is a subset of A such that there is an assignment of student a in class c in S.

The Student Sectioning Problem can be formulated as the standard Constraint Satisfaction Problem(CSP) in artificial intelligence. A CSP is a tuple $CSP = (V, U, W)$ with a set of variables V, domain set U, and a set of constraints W. A solution to a CSP is a set of assigment of values to variables with little or no violation of constraints. Thus, standard algorithms for solving CSP's, like backtracking, can be used to solve the Student Sectioning Problem.

In this paper, we present a multiagent system framework for solving the Student Sectioning problem. We model the student registration process as a multiagent system composed of autonomous agents that exhibits specific behavior to achieve their desired goals. The emergent interaction of the agents generate a solution to the Student Sectioning Problem. The main advantage of this approach is that the assignment can be done in parallel and in a distributed manner since each agent is autonomous having its own thread of execution and can be geographically dispersed.

2. METHODOLOGY
In this framework, we defined three types of agents namely *scheduler agent*, *enlister agent*, and *student agent*. These agents are representative of the actors that interact in the student registration process in a typical university. Agent communication is accomplished via *send*() and *receive*() primitives.

2.1 Scheduler Agent
The scheduler agent is the manager agent representative of the registrar. It bootstraps the enlister and student agents and responds to the queries from student agents (requesting initial schedules). It also collects the *schedule* from each student agent. Only one instance of the scheduler agent exists in the framework. The scheduler agent is responsible for collecting the final solution to the SSP.

Algorithm 1: Scheduler Agent Behavior

```
begin
    PercentCompleted ← 0;
    StartAllEnlisterAgents();
    StartAllStudentAgents();
    while PercentCompleted ≠ 100% do
        Message ← RECEIVE(StudentAgent);
        switch Message do
            case GET_INITIAL_SCHEDULE
                SEND(StudentAgent, Schedule);
            end
            case SUBMIT_FINAL_SCHEDULE
                UpdateFinalAssignment(StudentAgent,Schedule);
                UpdatePercentCompleted();
            end
        end
    end
    SENDTOALL(STOP);
    CommitFinalAssignment();
end
```

2.2 Enlister Agent

An enlister agent is responsible for responding to enlistment and cancellation requests from student agents. In the framework, each subject is assigned to an enlister agent. An enlister agent is responsible for enforcing the *full()* predicate as described in the problem definition of SSP.

Algorithm 2: Enlister Agent Behavior

```
begin
    Done ← false;
    GetAllClasslistsForSubject ();
    while Done ≠ true do
        Message ← RECEIVE(StudentAgent);
        switch Message do
            case GET_SECTIONS_WITH_SLOTS
                SEND(StudentAgent, SectionList);
            end
            case CANCEL_SLOT
                RemoveStudent(StudentAgent, Section);
            end
            case ENLIST_SLOT
                AddStudent(StudentAgent, Section);
            end
        end
        Message2 ← RECEIVE(SchedulerAgent);
        switch Message2 do
            case STOP
                Done = true;
            end
        end
    end
end
```

2.3 Student Agent

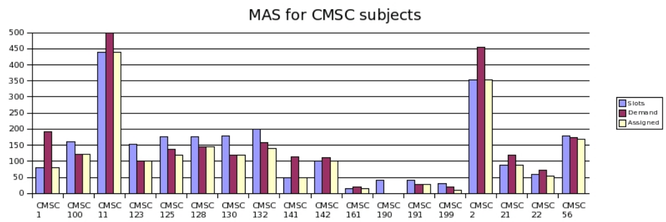
A student agent is responsible for obtaining an *assignment* and enforcing the *conflict()* predicate. Each student is represented by a student agent. A student agent has knowledge of a student's $write-in$ information which it uses to contact an enlister agent in an attempt to enlist.

Algorithm 3: Student Agent Behavior

```
begin
    Done ← false;
    SEND(SchedulerAgent,GET_INITIAL_SCHEDULE);
    WriteIn ← RECEIVE(SchedulerAgent);
    Schedule ← empty;
    while Done ≠ true do
        Subject ← SelectUnassignedSubject(WriteIn);
        SEND(EnlisterAgent(Subject),GET_SECTIONS_WITH_SLOTS);
        Sections ← RECEIVE(EnlisterAgent(Subject));
        if Sections is not empty then
            Section ←
            SelectNonConflictingSection(Sections);
            if Section not null then
                SEND(EnlisterAgent(Subject),ENLIST_SLOT,Section);
                AddToSchedule(Section);
                if Schedule is complete then
                    SEND(SchedulerAgent,SUBMIT_FINAL_SCHEDULE,
                    Schedule);
                    Done = true;
                end
            end
        end
        Message2 ← RECEIVE(SchedulerAgent);
        switch Message2 do
            case STOP
                Done = true;
            end
        end
    end
    SEND(SchedulerAgent,SUBMIT_FINAL_SCHEDULE,
    Schedule);
end
```

3. RESULTS AND DISCUSSION

A prototype implementation of the framework was developed using the Java Agent Development Environment (JADE)[1]. The figure below shows the the *available slots*, *demand*, and *assigned slots* using data from the authors' institute.



4. CONCLUSION

In this paper, we have presented a multiagent system framework that solves the Student Sectioning Problem. The multiagent approach advantage is that the finding of assignments can be done in parallel and in a distributed fashion.

5. REFERENCES

[1] F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi-agent systems with a FIPA-compliant agent framework. *Software: Practice and Experience*, 31(2), 2001.