*Jurnal Teknologi*, 46(D) Jun 2007: 45–60


# DYNAMIC TIMETABLING USING REACTIVE CONSTRAINT AGENTS

HANY ALASHWAL[1*] & SAFAAI DERIS[2]

**Abstract.** Most of the approaches that have been applied to solve the timetabling problems focus on the construction of the timetable as a static process. In real world, the timetabling problems are dynamic and open problems since the initial timetable is not fixed and it is required to be changed as the constraints or assumptions on which the timetable is based on, are changed or became invalid. Therefore, the main objective of this paper is to handle the changes after generating the initial timetable. The Reactive Constraint Agents (RCA) architecture is capable of repairing and modifying the timetable gradually by communicating and cooperating with each other to maintain the timetable feasibility. This architecture has been implemented and tested using real data from Faculty of Science, University of Ibb - Yemen. The results show that the RCA can cope with the changes in real-time with minimal modification to the existing timetable.

*Keywords:* Timetabling problem, dynamic timetabling, constraints programming, software agents, open agent architecture

**Abstrak.** Kebanyakan teknik yang diimplimentasi bagi menyelesaikan masalah penjadualan tertumpu kepada proses yang statik. Walau bagaimanapun, di dalam dunia sebenar, masalah penjadualan merupakan satu masalah yang terbuka, dinamik dan sentiasa berubah-ubah mengikut kekangan dan andaian. Oleh yang demikian, objektif utama kertas ini adalah untuk mengendalikan perubahan-perubahan yang berlaku setelah jadual waktu awalan terhasil. Agen Kekangan Reaktif (AKR) telah diimplimentasi lebih khusus dan berkeupayaan membaiki dan mengubahsuai jadual waktu secara bertahap dengan komunikasi dan kerjasama di antara satu sama lain bagi mengekalkan kesauran jadual waktu tersebut. Seni bina AKR ini telah dilaksana dan diuji dengan menggunakan data sebenar iaitu data dari Fakulti Sains, Universiti Ibb, Yemen. Hasil kajian menunjukkan bahawa AKR berupaya mengendalikan perubahan-perubahan dalam masa nyata dengan pembaikan yang minimum ke atas jadual waktu asal.

*Kata kunci:* Masalah penjadualan waktu, penjadualan waktu dinamik, pengaturcaraan terhad, agen perisian, seni bina agen terbuka

## 1.0 INTRODUCTION

The timetabling problems are combinatorial problems that consist of scheduling a set of courses within a given number of rooms and timeslots. Solving a real-world timetabling problem manually often requires a significant amount of time, sometimes

[1&2] Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia
* Corresponding author: Tel: +607 5532541, Fax: +607 5565044, Email: hany@siswa.utm.my

several days or even weeks. It has been known that the timetabling problem fall under the NP-complete problems [1]. Moreover, it is a dynamic and perturbed problem. Constraints alter as unexpected events occur, such as adding or deleting resources which are subjects or lecturers, and as the changes of the user demands put on to the scheduling system.

During the last thirty years, many contributions related to the timetabling problem have appeared and it will probably continue with the same rate for years. This could be due to the fact that timetabling problems are often over-constrained, dynamic, and optimization criteria are hard to define. Different techniques have and are being applied to solve the static timetabling problem, including graph coloring [2], integer programming [3] (from Operations Research), simulated annealing [4], tabu search [5], genetic algorithms [6], and constraint logic programming [7] (from Artificial Intelligence). Most of the existing timetabling systems focus on the static part of the timetabling problem and generate a near optimal solution. Moreover, the required modification or changes are usually done manually, which is difficult and time consuming.

In the dynamic timetabling problem, the main task is to minimally reconfigure schedules in response to a changing resources or activities [8]. A survey of current approaches to dynamic scheduling in general can be found in [9]. Dynamic timetabling problem has started to be investigated in [10]. Another approach to cope with the changes after the first schedule has been generated is by using an interactive tool [11]. Using this method, user must interact with the system to modify the schedule during the rescheduling process.

Recently, software agents have been applied to cope with the dynamic scheduling problems [12-14]. Agent Technology has received a great deal of attention among the researchers and practitioners in the field of Artificial Intelligence (AI). These agents are typically reactive, treating the world as an external memory from which knowledge can be retrieved by perception. Furthermore, the AI community has shown an increasing interest in solving Constraints Satisfaction Problems using the agent technology [15]. Constraint computation provides a general problem-solving framework whereas agents are self-directed problem-solving entities [16].

## 2.0 DYNAMIC TIMETABLING PROBLEM

In order to manage a rapid growth of academic activities in a university, an efficient and flexible timetabling must be developed. Figure 1 shows a typical university timetabling processes. Timetabling is thus an ongoing and continuous process. The problem of updating timetables in the most effective way, when the constraints or assumption on which they are based are changed or became invalid, is one that is receiving increasing attention amongst researchers and practitioners.

The main alternatives to the revision of a timetable are either by completely rescheduling the original timetable from scratch, or by repairing or modifying the

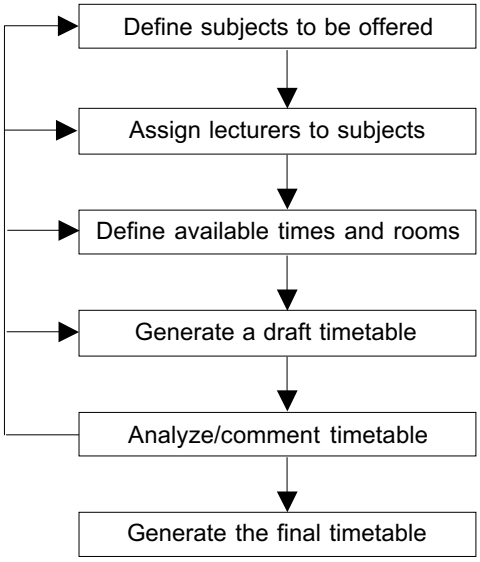


**Figure 1** Timetable construction process

previous timetable interactively such as the approach that has been used by [11]. However, most approaches to reactive scheduling are based on infrequent regeneration which cannot maintain continuity as it is progressively modified. Therefore, the aim of this paper is to show how to cope with the dynamic timetabling problem by using Reactive Constraint Agents (RCA). Many cases can arise which is always leading to some modifications to the current timetable such as:

(i) Turnover of academic staff.
   (a) Adding teaching staff: This problem can occur when teaching staff has just finished a study leave or new staff has joined. It can also be considered as a free staff.
   (b) Unplanned staff absences: This problem can occur due to retirement, illness, or emergence of other commitment of a teaching staff. It can also occur when a teaching staff gets a study leave.

(ii) Dynamic of enrolments make section or subject unavailable or inadequate.
   (a) Deleting sections or subjects. This problem occurs when there is not enough students to enroll in the subject. It can also be considered a free staff.
   (b) Adding new sections or subjects that previously are not being offered. This problem occurs in case of extra students or a new subject has been offered.
   (c) Adding/deleting activities (lectures, tutorials, seminars, etc.).

(d) Amending lecturers (swapped from one lecture to another, dropped from/ added to lecturers).
(e) Grouping teaching activities.

(iii) Request for changing timetable: This problem may occur due to lecturers' preferences to move class to better fit their timetable and so forth.

As shown above, it is difficult to maintain a given timetable on a real timetabling problem because of all kinds of disturbances that occur as mentioned above. The manual solution for the dynamic timetabling problem is based on the try-check principle. This manual technique is a time-consuming process. In addition, there is no guarantee that the new timetable is a conflict-free solution. Meanwhile, most of the current timetabling systems are static in which the first near optimal timetable is generated automatically. Then, any new timetable required to be computed due to changes or new requirements will be done manually.

## 3.0 REACTIVE CONSTRAINT AGENTS ARCHITECTURE

The Reactive Constraint Agents (RCA) is a multi-agent architecture aimed to implement a reactive system that is capable of coping with the dynamic timetabling problem. In the RCA, each agent executes specific types of tasks, and serves a specific purpose. No agent does an entire job. Rather, it does what it can, then delegates the other tasks to other agents. The RCA uses the Open Agent Architecture (OAA™) [17] as a platform. The Open Agent Architecture is a blackboard-based framework allowing individual software agents to communicate by means of goals posted on a blackboard controlled by a facilitator. The facilitator is responsible both for storing data that is global to the agents, identifying agents that can achieve various goals, and scheduling and maintaining the flow of communication during the computation. An extension of Prolog is used as the Inter-agent Communication Language (ICL) to take advantage of unification and backtracking when posting queries. The primary job of the facilitator is to decompose ICL expressions and route them to agents who have indicated a capability in resolving them. Thus, agents can communicate in an undirected fashion, with the blackboard acting as a broker. In the timetabling system, many activities have to be done in order to construct a timetable that satisfies all constraints simultaneously and optimize the timeslots and room as much as possible. By analyzing the system, the roles are identified and each role is mapped to an agent type. The agent should be able to perform the tasks associated with its role. In the timetabling process, there are different roles:

(i) Lecturer
(ii) Students' group

(iii) Administrators
(iv) Timetable designer

There are four basic domain agents that represent the described roles in the timetabling system. For these agents to do their tasks and communicate with each other, two other agents should be considered:

(v) Facilitator
(vi) Database agent

Figure 2 shows the architecture of the reactive constraint agents. This architecture consists of six types of agents:

(i) Facilitator: The OAA facilitator is a specialized server agent that is responsible for coordinating agent communications. The facilitator is also used to provide a global data store for its client agents, which allows them to adopt a blackboard style of interaction.

(ii) Administrator agent: This agent is created to represent the faculty administrator in the timetabling system. It has the authority to introduce a new change of the resources or the subjects in the timetabling system.

(iii) Timetabler agent: This agent is responsible for repairing the timetable when it is necessary. It can cooperate with the database agent, lecturer agents and students' group agents to accomplish his task.

(iv) Database agent: This is a special agent that establishes a connection with the timetabling database. It can provide the required data to the agent community.

(v) Lecturer agents: This is a personal agent to represent the lecturers in the timetabling system. It stores the availability and the preferences of its user. This agent can ask the timetabler agent to change his timetable. If there are available timeslots and rooms, the timetabler agent can change and update the timetable of the corresponding lecturer.

(vi) Students' group agents: This is a personal agent to represent the student groups in the timetabling system. It stores the availability and preferences of its user. This agent can ask the timetabler agent to change his timetable. If there is available timeslots-rooms and the related lecturer agent agree to change, the timetabler agent can change and update the timetable of the corresponding lesson.

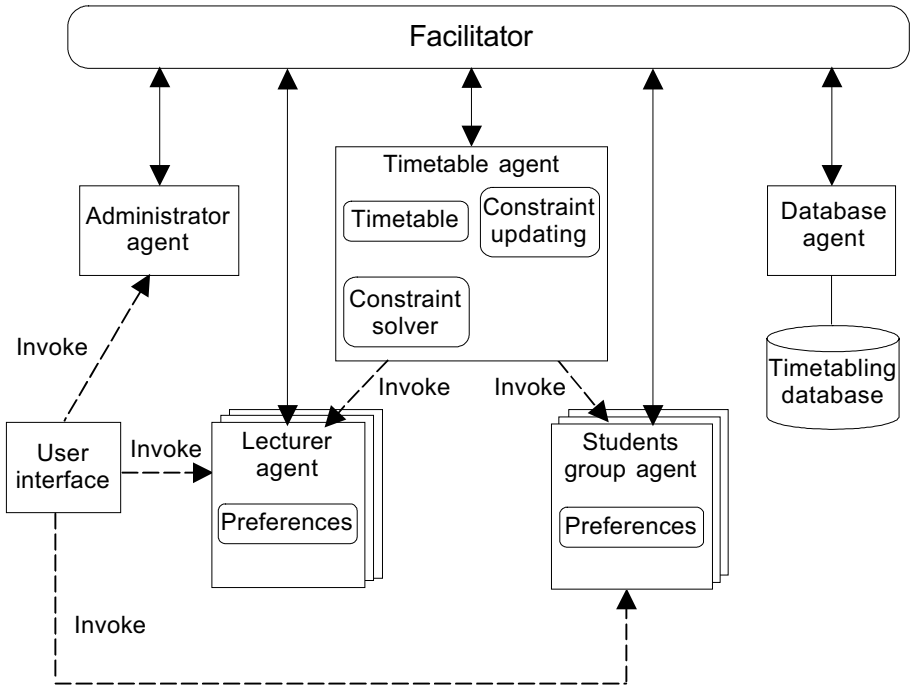


**Figure 2** The reactive constraint agents architecture

## 4.0 AGENTS COMMUNICATION AND COORDINATION

Communication among agents takes place through the facilitator. It usually does this by providing two main services: routing outgoing messages to the appropriate destination and translating incoming messages for consumption by its agents. Cooperation among the agents in the RCA is achieved via messages expressed in a common language; ICL, and is normally structured around a three-part approach: providers of services register capabilities specifications with a facilitator, requesters of services construct goals and relay them to a facilitator, and facilitators coordinate the efforts of the appropriate service providers in satisfying these goals.

When a new change is introduced, negotiation and cooperation among agents is necessary to resolve the constraints violation and repairing the existing timetable. As unexpected events or requests occur, negotiation and cooperation between agents is necessary to resolve the constraints violation and repairing the modified timetable. The agents interact with each other to recognize and categorize the conflict then select and apply the appropriate action in such way that all the constraints remain satisfied.

In the static timetabling problem, the constraints processing is done before the generation of the timetable (i.e. it is done during the assigning process), however in the dynamic problem, the constraint must be processed based on the current timetable.

This constraints processing can be regarded as real-time constraints processing. In order to handle the new events or the requested changes, the RCA uses algorithms that maintain the constraints in real time manner. In this problem, the variables are the timeslot $T(S_i)$ and the room $R(S_i)$ for each lesson $S_i$. The values to be assigned to timeslot variables $T(S_i)$ are the total available timeslots in a week, $t_j$, $1 \leq j \leq m$. The values assigned to a room variable $R(S_i)$ are the available rooms $rk$, $1 \leq k \leq p$, where $p$ is the number of the available rooms.

A solution to a timetabling problem can be defined as assignment of time $tj$, $1 \leq j \leq m$ and room $rk$, $1 \leq k \leq p$ to lessons $S_i$, $1 \leq i \leq n$ taught be lecturer $L(S_i)$ such that all constraints $C(S_i)$ are satisfied. $L(S_i)$ and $C(S_i)$ are lecturers and constraints of lesson $S_i$, respectively.

The constraints referred to the relationship between two variables. The basic constraints or relations are the mathematical relations, i.e., $\leq$, $\geq$, $=$ and $\neq$. The types of constraints that have to be satisfied in the timetabling process at Ibb University are as follows:

(i) Lecturer time-clash constraints: A lecturer cannot teach more than one subject in the same timeslot.

$$T(S_i) \neq T(S_j) \quad \text{if} \quad L(S_i) = L(S_j) \tag{1}$$

where $T(S_i)$ and $T(S_j)$ are the timeslots for the subject $S_i$ and $S_j$ respectively. $L(S_i)$ and $L(S_j)$ are the lecturers of the subjects $S_i$ and $S_j$ respectively, $i, j = 1, 2, ..., n$

(ii) Group time-clash constraints: A students group cannot attend more than one subject at the same timeslot.

$$T(S_i) \neq T(S_j) \quad \text{if} \quad G(S_i) = G(S_j) \tag{2}$$

where $G(S_i)$ and $G(S_j)$ are the students groups of the subjects $S_i$ and $S_j$ respectively, $i, j = 1, 2, ..., n$

(iii) Room time-clash constraints: Not more than one subject can be assigned to one room at the same timeslots.

$$T(S_i) \neq T(S_j) \quad \text{if} \quad R(S_i) = R(S_j) \tag{3}$$

where $R(S_i) = R(S_j)$ are the classrooms of the subjects $S_i$ and $S_j$ respectively, $i, j = 1, 2, ..., n$

(iv) Room capacity constraints: The number of students for subject assigned to the room must be less or equal to the capacity of the room.

$$N(S_i) \leq Z(R(S_i)) \tag{4}$$

where $N(S_i)$ is the number of students of the subject $S_i$ and $Z(R)$ is the capacity of the room $R$, $i, j = 1, 2, ..., n$

Before making any change, we must make sure that this change will not lead to violate any constraints. Therefore, we should check the lecturer time-clash constraints, group time-clash constraint, room time-clash constraint and room capacity constraint during the rescheduling process. The timetabler agent uses Get_Available algorithm (Figure 3) to find all the available timeslots and rooms for a specific lesson. Indeed to find the available timeslots and rooms, we must ensure that the change will not lead to violate any constraints.

```
Algorithm Get_Available
Input : timetable[n,3] : array of int //the first column represents the lesson
                                        id
                                      //the second column represents the
                                        timeslot
                                      //the third column represents the room
                                        id
                                      //n is the number of lessons
     subject[n]  : array of subject_record
     room[p]     : array of room_record //p is the number of rooms
     lessonId    : int
     no_of_timeslots: int
     continue    : boolean
Output : list_free_tm_rm
Begin
   k := 1
   continue := true
   While k < = n And continue Do
     If subject[k].lesson = lesson Then
        continue = false;
     Else k = k + 1
     EndIf
   EndWhile
   lecturer_id = subject[k].lecturer
   group_id = subject[k].group
   no_students = subject[k].no_students
   For timeslot = 1 To no_of_timeslots Do
     If is_lecturer_free(t, lecturerID) And is_group_free(t, groupID) Then
        For room = 1 To p Do
             If is_room_free(t,r) And room_capacity(r) >= no_Students Then
               list_free_tm_rm.Add(timeslot,room)
             EndIf
        EndFor
     EndIf
   EndFor
End
```

**Figure 3** The Get_Available algorithm

## 5.0 RESULTS AND DISCUSSION

To validate the results, it is essential to implement a system prototype of the proposed architecture to verify its properties. The prototype implementation is tested by modeling the timetabling problem at University of Ibb - Yemen. In this system prototype, the initial timetable was generated using the hybrid genetic algorithm which has been introduced in [18]. The timetabling problem at University of Ibb - Yemen has been modeled as a constraints satisfaction problem then solved using the hybrid genetic algorithm. The basic information in the timetable planning data is shown in Table 1. There are 129 subjects, 226 lessons and 16 rooms of various capacities. There are 18 timeslots (6 days a week with 3 timeslots per day of 3 hours per lesson).

**Table 1** Basic information of timetable planning data

| Items | Values |
|---|---|
| No. of subjects | 129 |
| No. of lessons | 226 |
| No. of teachers | 41 |
| No. of rooms | 16 |
| No. of timeslots | 18 |

The near-optimal timetable which was generated using the hybrid genetic algorithm is used as an initial timetable for the dynamic timetabling system (Figure 4). See Appendix A for the full timetable that includes all timeslots.

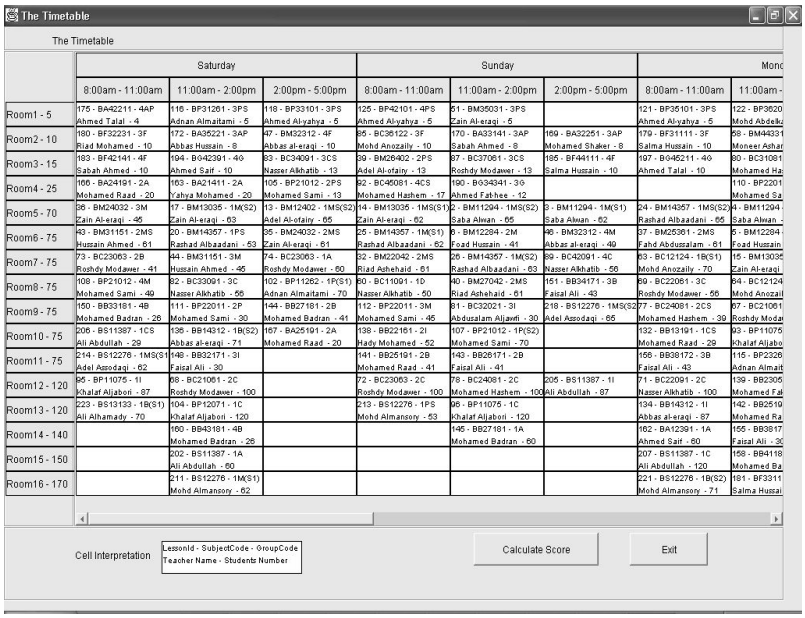


**Figure 4** The initial near-optimal timetable

For the system prototype implementation, we use Java programming language, and for the database, we use Microsoft Access. As an example, we trace the scenario of deleting a classroom which leads to reschedule all the lessons that are assigned to this classroom. The following is an example of an operational demonstration scenario that illustrates inter-agent communication (Figure 5).

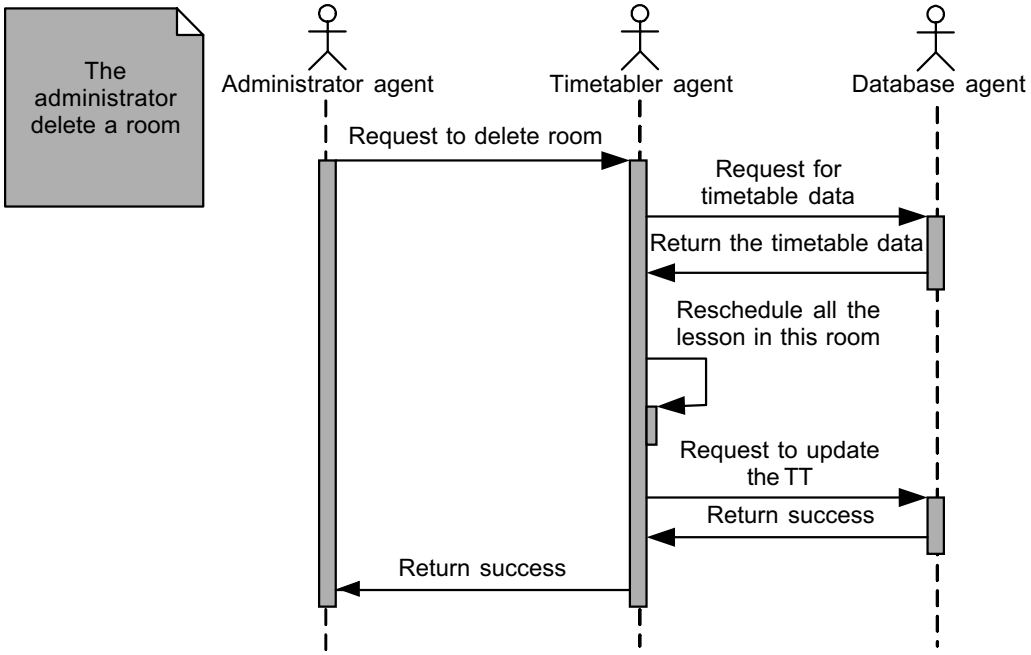


**Figure 5** Example of agent interaction

When the administrator agent send a request for deleting a classroom, the timetabler agent receive this request and try to satisfy it. In order to handle this request, the timetabler agent need to cooperate with the database agent to get the sufficient timetabling data. After the timetabler agent gets the sufficient timetabling data, it applies appropriate action to reschedule all the lessons that were assigned in the deleted room. The new timetable is feasible and has minor changes from the initial timetable (see Figure 6). See Appendix A for the full new timetable that includes all timeslots. It also shows the difference between the initial timetable and the new timetable after deleting Room 16. For different scenarios that lead to changing timetable, see [19].

Unlike the previous interactive approach to cope with the dynamic timetabling problem that has been reported in [11], the use of reactive constraints agents which is introduced in this research, relieve human user from the responsibility of interfacing, task planning, and execution monitoring. This has several benefits,

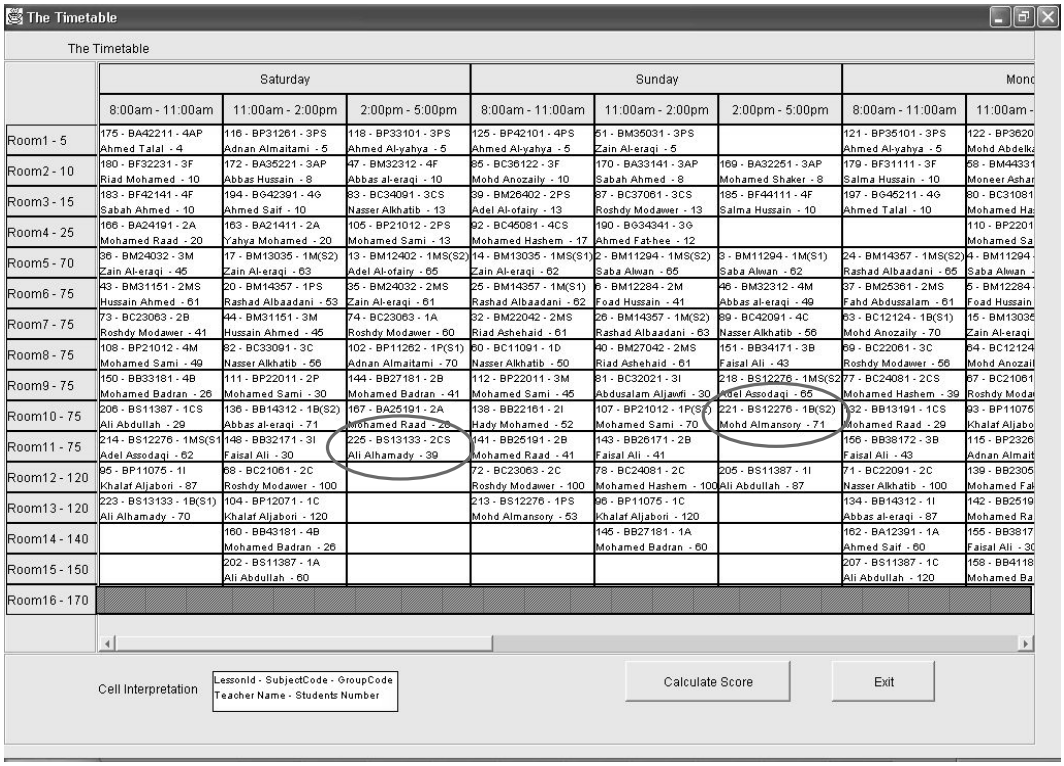


**Figure 6** The timetable after deleting Room 16

including reducing the complexity for users and agents, precipitating a more open and dynamically extensible computing style, and encouraging reuse across applications and domains.

Recently, [10] have used constraint-programming-based tools for solving dynamic timetabling problems modelled as resource-constrained project scheduling problems. Comparing this approach with the agent technology approach, the reactive constraint agents' architecture provides these advantages:

(i) Flexibility of the system.
(ii) Extensibility of the architecture, making it particularly easy to add and enhance agents.
(iii) Modularity in which each agent is independent which eases development and maintenance.
(iv) Ease of integration with other systems through the encapsulation of existing problem-solving systems as agents.
(v) Reusability of the agents across multiple domains (e.g. utilizing existing agents in different scheduling systems).
(vi) Adaptable to distributed environment.

**Table 2** The processing time for agents' tasks

| Task | Agent | Time |
|---|---|---|
| Get available timeslots and rooms | Timetabler agent | 8 seconds |
| Reschedule all lessons in a room | Timetabler agent | 56 seconds |
| Add lesson | Timetabler agent | 6 seconds |
| Provide the timetable data | Database agent | 5 seconds |
| Update the timetable data | Database agent | 2 seconds |

The time performance of the system is shown in Table 2 which shows that the agents can react to the events in the timetabling system in a real time manner. In contrast, the manual modification of the initial timetable may take several days or even weeks.

In addition to the advantages of using agents' technology to handle the dynamic timetabling problem, the proposed reactive constraint agents' architecture can be applied to other dynamic problems, like dynamic manufacturing scheduling problem, meeting scheduling problem, and staff scheduling problem, with minor modification. This is due to the modularity of the agents and the flexibility of the system.

## 6.0 CONCLUSION AND FUTURE WORK

In this paper, we have presented a reactive constraint agents architecture that is capable of coping with the dynamic timetabling problem. The architecture has been implemented and a prototype has been produced. The implementation has been tested using real data from University of Ibb - Yemen. The experiments show that when a change is required to be done on the existing timetable, the timetabler agent can cooperate with other agents in the system to modify the timetable in such a way that all the constraints are satisfied simultaneously. Future work is needed to fully test all type of changes that can occur in the timetabling environment. Furthermore, the architecture can be implemented in a distributed environment.

## REFERENCES

[1] Tim, B. C. and J. H. Kingston. 1995. The Complexity of Timetable Construction Problems. In Proceedings of the 1st International Conference on Practice and Theory of Automated Timetabling (PATAT 1995). LNCS 1153. Springer-Verlag. 283-295.

[2] Miner, S., S. Elmohamed, and H. W. Yau. 1995. Optimizing Timetabling Solutions Using Graph Coloring. NPAC REU Program, NPAC. Syracuse University. Syracuse, NY.

[3] Dimopoulou, M. and P. Miliotis. 2001. Theory and Methodology Implementation of a University Course and Examination Timetabling System. *European Journal of Operational Research.* 130: 202-213.

[4] Melício, F., P. Caldeira, and A. Rosa. 1999. Solving the Timetabling Problem with Simulated Annealing. Proc. First Int. Conf. on Enterprise Information Systems, ICEIS' 99 - Setúbal. 272-279.

[5] Schaerf, A. 1996. Tabu Search Techniques for Large High-school Timetabling Problems. In Proceedings of the Fourteenth National Conference on Artificial Intelligence. Portland, Oregon. 363-368.

[6] Colorni, A., M. Dorigo, and V. Maniezzo. 1990. Genetic Algorithms: A New Approach to the Time Table Problem. NATO-ASI School on Combinatorial Optimization. Ankara, Turkey.

[7] Azevedo, F. and P. Barahona. 1994. Timetabling in Constraint Logic Programming. In Proceedings of 2nd World Congress on Expert Systems. Estoril, Portugal.

[8] El Sakkout, H. and M. Wallace. 2000. Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling. *CONSTRAINTS*. 4(5): 359-388.

[9] Kocjan, W. 2002. Dynamic Scheduling: State of the Art Report. Technical Report T2002:28. SICS.

[10] Elkhyari, A., C. Guéret, and N. Jussien. 2003. Solving Dynamic Timetabling Problems as Dynamic Resource Constrained Project Scheduling Problems Using New Constraint Programming Tools. In *Practice and Theory of Automated Timetabling, Selected Revised Papers*. Springer-Verlag LNCS 2740. 39-59.

[11] Cambazard, H., F. Demazeau, N. Jussien, and P. David. 2004. Interactively Solving School Timetabling Problems Using Extensions of Constraint Programming. *Practice and Theory of Automated Timetabling V*. LNCS, Springer-Verlag. 190-207.

[12] Kouiss, K., H. Pierreval, and N. Mebarki. 1997. Using Multi-Agent Architecture in FMS for Dynamic Scheduling. *Journal of Intelligent Manufacturing*. 8(1): 41-47.

[13] Gozzi, A., M. Paolucci, and A. Boccalatte. 2002. A Multi-Agent Approach to Support Dynamic Scheduling Decisions. In Proc. of ISCC2002. Taormina, Italy. 983-988.

[14] Ouelhadj, D. 2003. Multi-agent based Scheduling. In the Workshop of Open Issues in Grid Scheduling. National e-Science Centre, Edinburgh.

[15] Shen, W. 2002. Distributed Manufacturing Scheduling Using Intelligent Agents. *IEEE Intelligent Systems*. 17(1): 88-94.

[16] Eaton, P. S., E. C. Freuder, and R. J. Wallace. 1998. Constraints and Agents: Confronting Ignorance. *AI Magazine*. 19(2): 51-65.

[17] Martin, D. L., A. J. Cheyer, and D. B. Moran. 1999. The Open Agent Architecture: A Framework for Building Distributed Software Systems. *Applied Artificial Intelligence*. 13(1/2): 91-128.

[18] Deris, S., S. Omatu, H. Ohta, and P. Saad. 1999. Incorporating Constraint Propagation in Genetic Algorithm for University Timetable Planning. *Journal of the Engineering Application of Artificial Intelligence*. 12: 241-253.

[19] Alashwal, H. 2003. The Development of Reactive Constraint Agents for the Dynamic Timetabling Problem. M.Sc. Thesis. Universiti Teknologi Malaysia.

# APPENDIX A

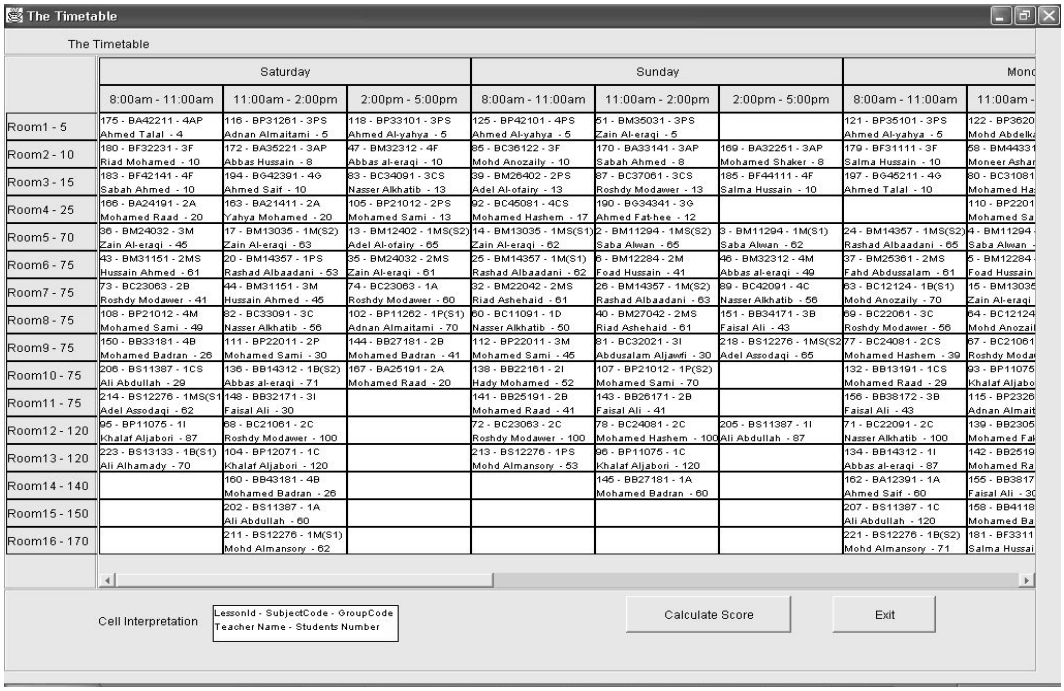


**Figure A1a** The initial timetable with the first timeslots

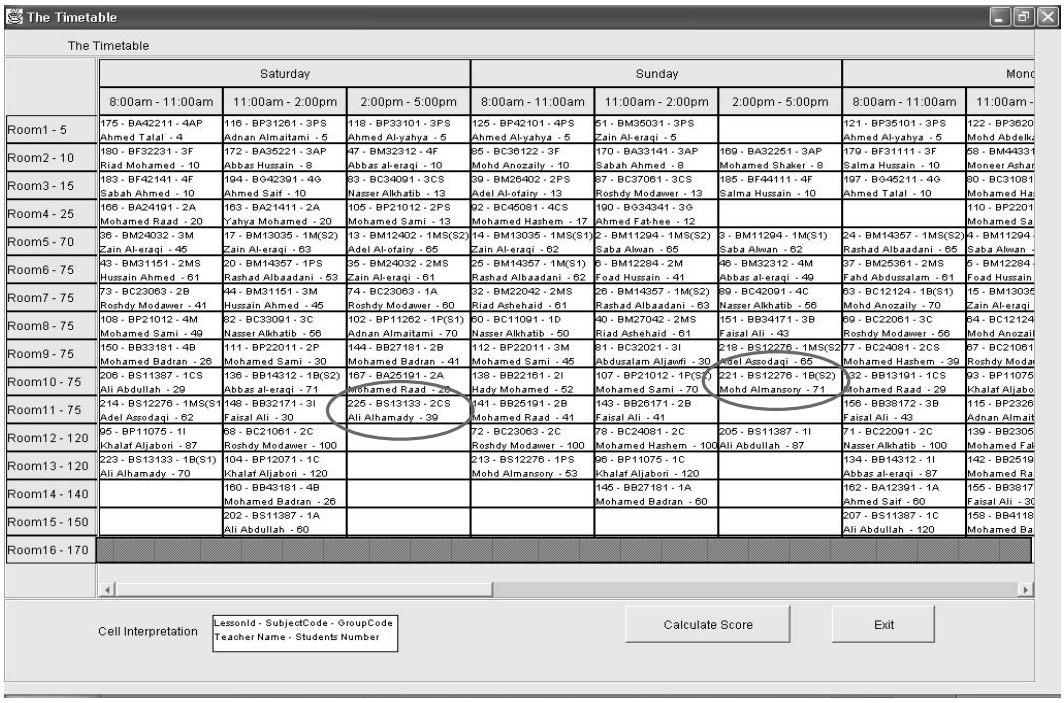


**Figure A1b** The new timetable with the first timeslots

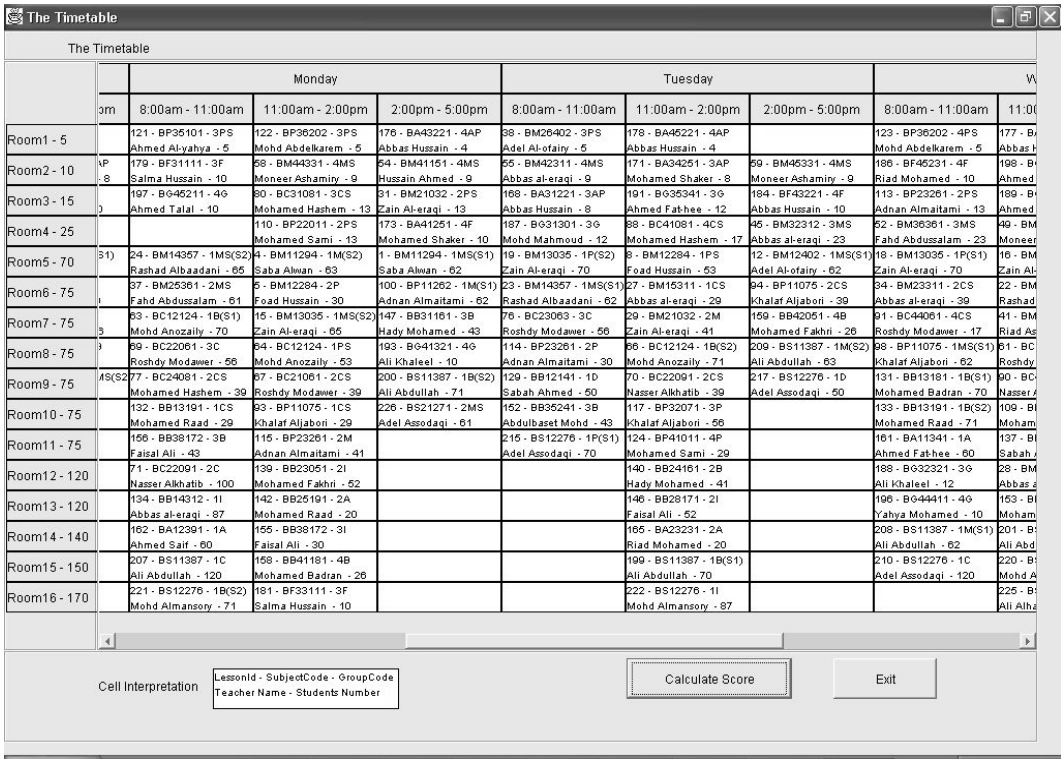


**Figure A2a** The initial timetable with the middle timeslots

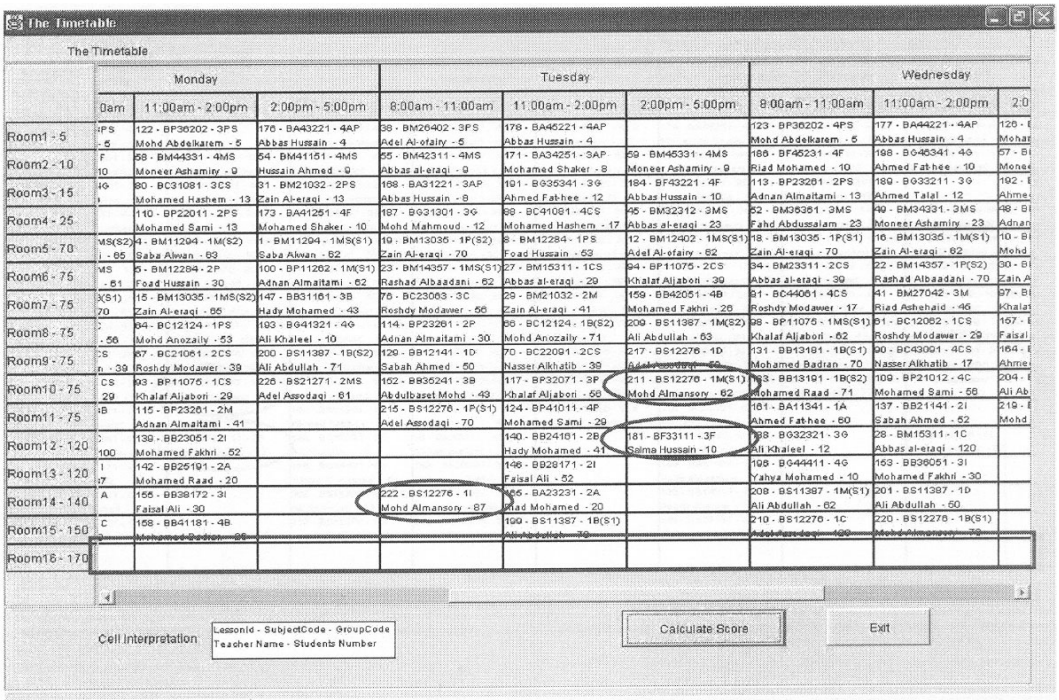


**Figure A2b** The new timetable with the middle timeslots

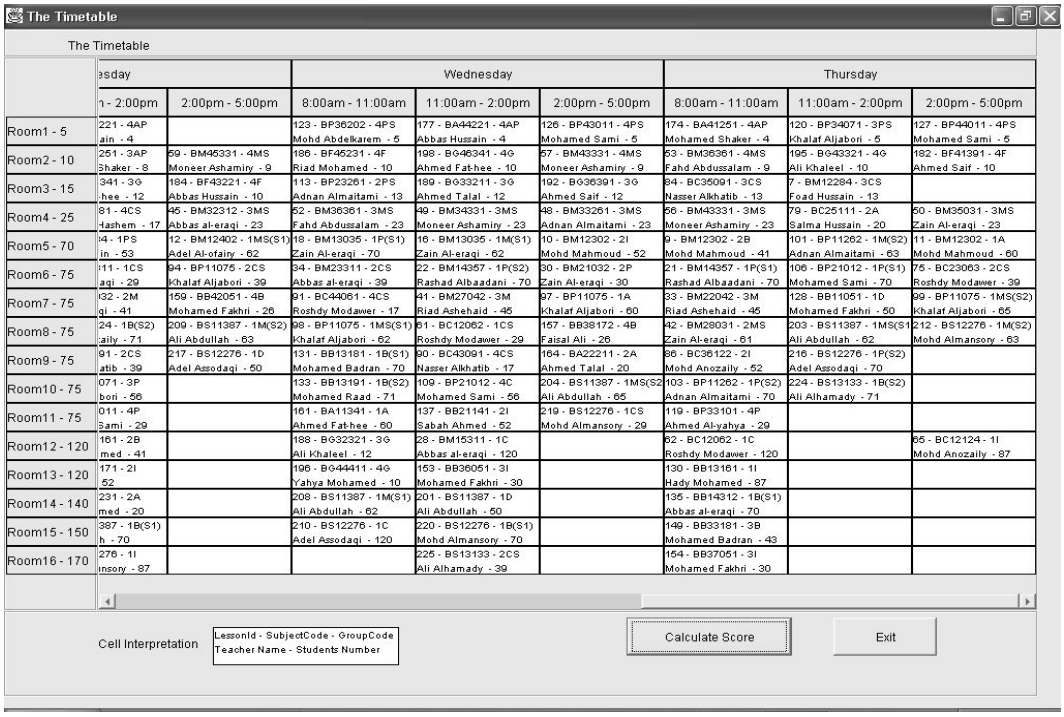


**Figure A3a** The initial timetable with the last timeslots

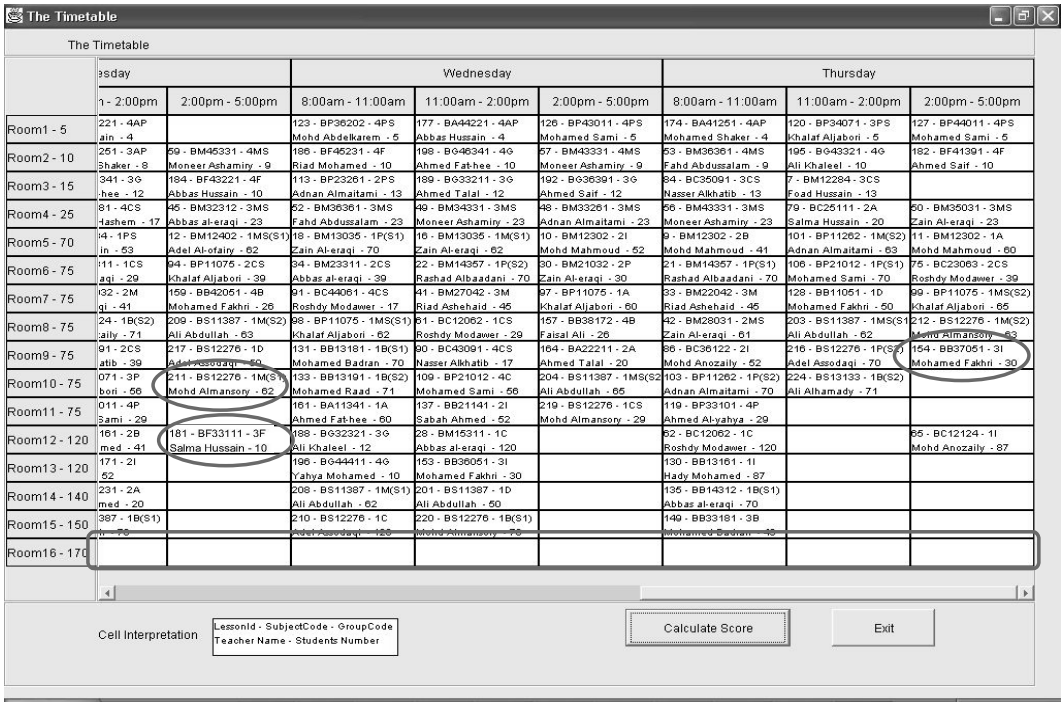


**Figure A3b** The new timetable with the last timeslots