SchoolTimetable: an hybrid CP and MAS case study

Giuseppe Boezio
University of Bologna
giuseppe.boezio@studio.unibo.it

Abstract

Agent-oriented programming can be very useful to model and solve real life complex problems involving interaction among different individuals. In this paper it is described the problem of school timetable. For this problem a model based on Multi-Agent System results to be suitable for describing the way individuals could interact. Moreover, the Constraint Programming paradigm is used in addition to the aforementioned paradigm to allow to solve successfully a problem where constraints over the final solutions must be satisfied.

1 Introduction

Agent-oriented programming is a paradigm which is based on the concept of software agents. Several definitions have been provided, one of the most used is described in [Wooldridge (2009)] and claims that an agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives. In most of the contexts there is more than a single agent in an environment and therefore it is possible to define complex distributed systems called Multi-Agent Systems (MAS) as extension of the agent technology where a group of loosely connected autonomous agents act in an environment to achieve a common goal. This is done either by cooperating or competing, sharing or not sharing knowledge with each other [Parasumanna Gokulan and Srinivasan (2010)].

MAS are useful in contexts where a problem cannot be solved by a single agent but for several reasons (e.g. scalability, performance, privacy, etc...) an interaction among agents is required. One of these contexts is represented by problems where there are constraints among elements which represent the solution. This kind of problems is commonly solved using a paradigm called Constraint Programming (CP). Constraint Programming is a paradigm for solvig combinatorial problems where different constraints are imposed on feasible solutions for different decision variables, each having its own domain. Constraints are relations among variables which limit the values decision variables can assume in feasible solutions [Rossi et al. (2006)]. If every possible solution which satisfies the constraints is accepted, the problem is a Constraint Satisfaction Problem (CSP) otherwise, if in addition to the constraints the solution must maximize (or minimize) a certain function, the problem is a Constraint Optimization Problem (COP). Constraint Programming can also be used together with Logic Programming; in this case it is called Constraint Logic Programming (CLP) [Jaffar and Maher (1994)].

Constraint problems often involve multiple participants even if the problem can be solved by gathering all constraints and optimization criteria into a single large CSP, and then solving this problem using a centralized algorithm. In some conditions this kind of approach is unfeasible for several reasons (cost of formalization, privacy, dynamicity and brittleness); therefore a new distributed constraint programming approach is used. This is called Distributed Constraint Programming (DisCP) and differs from Constraint Programming because each variable is controlled by an agent (meaning that the agent sets the variable's value). Each agent can control more than one variable; in this case all variables set by an agent are modelled as a single variable [Rossi et al. (2006)].

Preprint. Under review.

In this paper it is described a problem which involves both Constraint Programming and Multi-Agent Systems. The problem consists of finding a school timetable for each professor of a school such that different constraints related to different aspects of the problem are respected, and trying to satisfy all preferences of each professor as much as possible.

2 Requirements

Italian schools have a person (usually a professor) who is responsible for the creation of an overall timetable where for each day of the week and for each hour, a professor must be assigned to a class according to the hours that must spend in each class as described by school regulation. This task is difficult and time-demanding because it is subjected to different constraints related to professors and classes where they teach. Sometimes it can also happen that a professor asks for a lesson change due to own commitments or needs and this entails finding a lesson change which keeps still valid the aforementioned constraints.

More in details the requirements are the followings:

- There is a person, responsible for the creation of the timetable of each professor (time-scheduler), who receives from school direction all information to perform the aforementioned task
- After the creation of timetables, the time-scheduler must send to each professor the corresponding timetable
- Each professor must spend exactly in each class a number of hours described by the school regulation according to the number and type of class where the teaching happens
- Each professor must have a free day during the school week
- In the same class cannot be more than one professor per hour (this requirement has been imposed to simplify particular cases, e.g. laboratories, where two professor teach in the same class)
- Each professor can have own preferences related to the lessons in which he/she would rather not teach
- Each professor can begin a negotiation trying to satisfy own preferences, mediated by the time-scheduler

3 Implementation

The system has been implemented using a MAS framework called Jade [Bellifemine et al. (2001)] and a custom implementation of a CLP library which shares the same predicate interfaces of SWI Prolog interpreter [Wielemaker (1987)].

Jade (Java Agent DEvelopment framework) is a MAS framework developed in Java which supports the notion of agent. It has been used to simulate the time-scheduler and the professors as agents which interact in the same environment.

The custom CLP library has been developed in Kotlin for the Prolog interpreter 2p-Kt [Ciatto et al. (2021)] and it is used to satisfy the constraints related to the formulation of timetables.

4 Design

Several design choices have been taken and will be explained according to the kind of paradigm they affect.

4.1 Multi-Agent Systems (MAS)

4.1.1 Agents and Behaviours

There are two type of agents: **TimeScheduler** and **Professor**.

TimeScheduler agent has in its own state all information related to the amount of hours each professor must spend in each class and knows the Agent Identifier (AID) of each **Professor** agent. Moreover, it uses the Directory Facilitator (DF) agent to register its own service of negotiation mediator.

The **TimesScheduler** is responsible for the following tasks:

- creation of timetables
- communication of timetables to each professor
- mediation of negotiation among professors for the satisfaction of their preferences

These tasks are accomplished with the following Behaviors:

- **TimetableBehaviour**: it produces the different timetables, save and send each of them to the corresponding professor
- **WaitProposalBehavior**: it waits for a cfp message from a professor to begin a negotiation
- **MediationBehaviour**: it mediates negotiation among professors for preferences satisfaction

Professor agent is an abstract class which is used to keep a common architecture for all professors which differ only for their preferences.

Professor agents perform the following tasks:

- receive the timetable
- try to satisfy own preferences
- decide whether to accept a lesson change or not

In order to do these tasks they have the following Behaviours:

- **TimetableBehaviour**: it receives the timetable and save it
- **PreferenceBehaviour**: for a specific number of times it tries to satisfy unsatisfied preferences
- **NegotiationBehaviour**: it starts a negotiation for a specific unsatisfied preference
- **WaitProposalBehavior**: it waits for a request message from the timescheduler corresponding to a change proposal and manages it with a CandidateBehaviour
- **CandidateBehaviour**: it replies to the proposal of a specific lesson change

Professor agents, after receiving the timetable, register the classes where they teach as services interacting with the DF agent. This is done because when the **Timescheduler** agent will receive a lesson change request, it will look for all professors who teach in the same class where the proposed change happens.

Another important design choice is related to the following aspect: when a professor asks for a lesson change, it must be sure that before the end of the negotiation, the current lesson cannot be available to other incoming requests. To do this, in the **NegotiationBehaviour** the current lesson is added to a set of locked lessons. In this way if another professors ask him/her to change this lesson, the request will not be able to be satisfied.

4.1.2 Communication among agents

The interaction among time-scheduler and professors can be explained as follows (as shown in figure 1):

when a professor wants to change an own lesson in order to satisfy a preference, a call-for-proposal (cfp) message is sent to the time-scheduler to begin the negotiation. The time-scheduler will retrieve the list of all professors who teach in the same class of the proposer's lesson. To have a reasonable change it is also important to consider that for each candidate professor and for each candidate lesson:

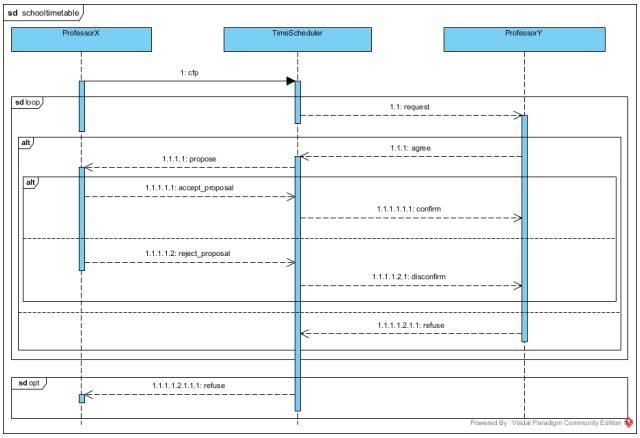

Figure 1: Sequence Diagram of the interaction among agents

- the proposer must be free during a candidate lesson
- the candidate professor must be free during the lesson the time-scheduler wants to propose.
- the proposer's lesson cannot be in the free day of the candidate professor and the candidate lesson cannot be in the free day of the proposer professor

Following these criteria, the time-scheduler will realize a list of pair professor-lesson and will try to satisfy the proposer's request with one of them.

The time-scheduler sends to the candidate professor a proposal and this proposal can be accepted according to two criteria:

- the proposed lesson is not one of own preferences
- the lesson to give must not be in the locked preferences

If these criteria are met, the candidate professor replies with an agree message otherwise refuses.

If the reply message contains agree as performative, the time-scheduler communicates to the proposer the possible change. The proposer will accept or reject according to the fact that the lesson change does not worsen own preferences.

According to the last message received from the proposer, the time-scheduler will send to the candidate, who previously accepted the change, a confirm or disconfirm message.

If all possible negotiations fail, the time-scheduler will send to the proposer a refuse message.

4.1.3 Ontology

As described in [Bellifemine et al. (2001)] the content of a message is either a string or a raw sequence of bytes. In realistic case, like in this one, agents need to communicate complex information. When representing complex information, it is necessary to adopt a well-defined syntax so that the content of a message can be parsed by the receiver to extract each specific piece of information. According to FIPA terminology this syntax is known as a content language. FIPA does not mandate a specific content language but defines and recommends the SL language to be used when communicating with the AMS and DF.

For this reason the SL language is used as content language for all messages involving the time-scheduler and the professors. The ontology shared by all agents consists of four concepts and three predicates. The different elements are described with SL syntax as follows:

Concepts

$$(Lesson : hour\langle hour\rangle : day\langle day\rangle)$$
$$(SchoolClass : year\langle year\rangle : letter\langle letter\rangle)$$

$$(Teaching : lesson\langle hour\rangle : schoolClass\langle class\rangle)$$

$$(TimetableConcept : teachings\langle list\ of\ teachings\rangle)$$

Predicates

$$(UpdateTimetable : timetable\langle timetable\ of\ an\ agent\rangle)$$

$$(Change : lessonChange\langle lesson\ related\ to\ a\ change\rangle)$$

$$(Substitution : proposedLesson\langle proposer's\ lesson\rangle : currentLesson\langle candidate's\ lesson\rangle)$$

TimetableConcept is used to encode as content message the timetable of each professor stored as an object of the class **Timetable**. The reason behind two different classes is due to the fact that **Timetable** uses a matrix to store information about lessons and it is faster to retrieve and add elements compared to the class **TimetableConcept**. On the other hand SL language does not support the matrix concept and this is why a concept modelled as a list of teachings is needed.

4.2 Constraint Programming (CP)

The CP model is realized using the aforementioned 2p-Kt interpreter and in particular the **PrimitiveWrapper** class which allows to define predicates that generate a side effect without involving the standard unification process.

In this way it was possible to map the predicate interface of the clpfd SWI Prolog library [Triska (2012)] with a Java based constraint programming library called Choco [Prud'homme and Fages (2022)].

In the following part it is provided a mathematical formulation of the model and the corresponding code in Prolog.

The model should be generated dynamically according to the single instance data but to speed up the development process, the current version of this project considers a specific instance of the problem.

4.2.1 Data provided by the problem

Days of the school week

$$numDays \in \{1..6\}$$

Hours for each school day

$$numHours \in \{1..24\}$$

Number of different professors

$$numProfessors \in \mathbb{N}^+$$

Number of different classes

$$numClasses \in \mathbb{N}^+$$

Function which assigns for each professor and each class the number of hours:

$$\forall i \in \{1..numProfessors\}$$

$$hours_i : \{1..numClasses\} \rightarrow \{0..(numHours * numDays)\}$$

4.2.2 Domain of variables

$$\forall i \in \{1..numProfessors\}, j \in \{1..numHours\}, k \in \{1..numDays\}$$

PROLOG

$$P_{ijk}\ in\ 0..numClasses$$

NOTE: 0 is used as joker value to state no class

4.2.3 Constraints

Each Professor must teach for a specified number of hours in each class:

$$\forall i \in \{1..numProfessors\}$$

PROLOG

$global\_cardinality([P_{i**}], [1 - hours_i(1), 2 - hours_i(2), .., numClass - hours_i(numClass)])$

NOTE: the number of same values assigned to professor's variables must be equals to the number of hours the professor must spend in that class.

$[P_{i**}]$ means the list of variables associated to professor i

In the same day and hour cannot be more than one professor in each class

$$\forall j \in \{1..numHours\}, k \in \{1..numDays\}$$

PROLOG

$$all\_distinct\_except\_0([P_{*jk}])$$

NOTE: $[P_{*jk}]$ is the list of variables associated to all professors having the same day and hour.

Each Professor must have a free day

$$\forall i \in \{1..numProfessors\}$$

PROLOG

$$tuples\_in([P_{i*1}], [[0..0]]) \ \vee \ tuples\_in([P_{i*2}], [[0..0]]) \ \vee \ .. \ tuples\_in([P_{i*(numDays)}], [[0..0]])$$

NOTE: $[P_{i*1}]$ is the list of all variables of professor i at day 1 for all hours.

4.2.4 Performance

The solution of the CP model requires a lot of time to be computed because of the huge amount of variables and constraints imposed on them. Therefore a **DummyBehaviour** has been realized to simulate the creation of the professors' timetable and then each timetable is sent to the corresponding professor.

5 Conclusions and future work

This project shows how in some contexts it is not possible to use a single paradigm but different ones must be adopted to exploit on respective strengths in order to solve a complex problem like timetable scheduling.

Some aspect of the problem have been simplified, therefore the following new features could be implemented:

- information provided by school direction could be stored in files or database instead of storing it in the program
- more complex constraints could be added to make the scenario more realistic than it is. For instance, each class could have a different amount of hours per day or more professor could teach during the same lesson in a class.
- a different model could be developed to improve the performance issue
- the CP model could be dynamically created to allow a more general and flexible approach to solve this problem
- a graphical user interface could be realized to have a better understanding of timetable and hour change

References

Bellifemine, F., Poggi, A., and Rimassa, G. (2001). Developing multi-agent systems with jade. In Castelfranchi, C. and Lespérance, Y., editors, *Intelligent Agents VII Agent Theories Architectures and Languages*, pages 89–103, Berlin, Heidelberg. Springer Berlin Heidelberg.

Ciatto, G., Calegari, R., and Omicini, A. (2021). 2p-kt: A logic-based ecosystem for symbolic ai. *SoftwareX*, 16:100817.

Jaffar, J. and Maher, M. J. (1994). Constraint logic programming: a survey. *The Journal of Logic Programming*, 19-20:503–581. Special Issue: Ten Years of Logic Programming.

Parasumanna Gokulan, B. and Srinivasan, D. (2010). *An Introduction to Multi-Agent Systems*, volume 310, pages 1–27.

Prud'homme, C. and Fages, J.-G. (2022). Choco-solver: A java library for constraint programming. *Journal of Open Source Software*, 7(78):4708.

Rossi, F., van Beek, P., and Walsh, T. (2006). *Handbook of Constraint Programming*. Elsevier Science Inc., USA.

Triska, M. (2012). The finite domain constraint solver of SWI-Prolog. In *FLOPS*, volume 7294 of *LNCS*, pages 307–316.

Wielemaker, J. (1987). Swi prolog.

Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition.