A Language for Specifying Complete Timetabling Problems

Luís Paulo Reis[1,2], Eugénio Oliveira[1,3]

lpreis@ufp.pt, eco@fe.up.pt

[1]LIACC – Artificial Intelligence and Computer Science Lab. – University of Porto, Portugal
Http://www.ncc.up.pt/liacc/, Tel: 351-22-5081315, Fax: 351-22-5081315

[2]UFP - CEREM – Multimedia Resource Center, Praça 9 de Abril, 349, 4200 Porto, Portugal

[3]FEUP – DEEC, Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal

Abstract. The timetabling problem consists in fixing a sequence of meetings between teachers and students in a given period of time, satisfying a set of different constraints. There are a number of different versions of the timetabling problem. These include school timetabling (where students are grouped in classes with similar degree plans), university timetabling (where students are considered individually) and examination timetabling (i.e. scheduling of university exams, avoiding student double booking). Several other problems are also associated with the more general timetabling problem, including room allocation, meeting scheduling, staff allocation and invigilator assignment. Many data formats have been developed for representing different timetabling problems. The variety of data formats currently in use, and the diversity of existing timetabling problems makes the comparison of research results and exchange of data concerning real problems extremely difficult.
In this paper we identify eight timetabling sub-problems and, based on that identification, we present a new language (UniLang) for representing timetabling problems. UniLang intends to be a standard suitable as input language for any timetabling system. It enables a clear and natural representation of data, constraints, quality measures and solutions for different timetabling (as well as related) problems, such as school timetabling, university timetabling and examination scheduling.

1 Introduction

Wren [46] defines timetabling as a special case of scheduling: “Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space-time, in such a way as to satisfy as nearly as possible a set of desirable objectives”. In a more common definition, timetabling problem consists in fixing in time and space, a sequence of meetings between teachers and students, in a prefixed period of time, satisfying a set of constraints of several different kinds. These constraints may include both hard constraints that must be respected and soft constraints used to evaluate the solution quality.

The scientific community has given a considerable amount of attention to automated timetabling during the last four decades. Starting with the works of Appleby [2] and Gotlieb [25], many papers have been published in conferences and journals, including several surveys [3,7,9,11,12,19,30] and annotated bibliographies

on the same subject [33,41]. In addition, several practical timetabling systems have been developed and applied with partial success. The first automated timetabling approaches were based on operational research methodologies like network flow techniques [20], reduction to graph colouring [45], integer programming [35], direct heuristics [4], simulated annealing [1], tabu search [29] and neural networks [34]. Despite timetabling automation may be desirable, there are also a number of clear advantages of manual timetabling over automated timetabling. This lead several authors [44] to advocate the use of interactive methods for timetabling based on decision support systems and human-computer interfaces techniques. In more recent years, research in automated timetabling includes techniques like logic programming [23,31], expert systems [26], constraint logic programming [27,37,39,42] and genetic/evolutionary algorithms [6,17].

A large number of variants of the timetabling problem have been proposed in the literature. Schaerf [40] classifies the timetabling problem in three main classes based on the type of institution involved and the type of constraints:

- **School Timetabling** - Weekly scheduling for all the classes of a high school avoiding teachers and groups of student double booking;
- **University Timetabling** - Scheduling of all the lectures of a set of university degree modules, minimising the overlaps of lectures having common students and avoiding teachers double booking;
- **Examination Timetabling** - Scheduling the exams of a set of university courses avoiding overlapping exams for the same students and spreading the exams for the students as much as possible.

As it was noticed by Schaerf [40], this classification is not strict in the sense that some specific problems may fall in between two classes and cannot be easily placed in this classification. Carter, in his study about recent developments in practical course timetabling [11] identified five different sub-problems for the course scheduling problem: course timetabling, class teacher timetabling, student scheduling, teacher assignment and classroom assignment.

The recent emergence of the PATAT – Practice and Theory of Automated Timetabling series of international conferences [5,8] and the establishment of the EURO (Association of European Operational Research Societies) Working Group on Automated Timetabling (WATT), indicates that the research interest in this area is increasing dramatically. Still, the variety of data formats currently in use and the diversity of existing timetabling problems makes the comparison of research results and exchange of research ideas and data concerning real problems extremely difficult. Several attempts towards finding a standard to represent timetabling problems were made in the past but at the present there is no universally accepted language for describing timetabling problems.

This paper introduces UniLang, a new language for representing timetabling problems that can be easily read by computer specialists and school administrators. In Section 2, some timetabling data standards proposed in the past years are briefly reviewed. Eight sub-problems identified in the complete timetabling problem are introduced in Section 3 along with the architecture for a generic timetabling system. Section 4 presents the requirements for a language to represent timetabling problems. Based on these requirements as well as on the timetabling related sub-problems, we present in Section 5 our proposal of a language for representing timetabling problems. Section 6 briefly shows how a problem represented through this language can be

translated to a constraint logic program and solved using a constraint logic programming language. Finally, we give some conclusions together with an outlook to future research.

2 Timetabling Data Standards

At the present there is no universally accepted language for describing timetabling problems. Several attempts were made towards finding a standard information format for timetabling problems and although some data formats have been developed for representing different timetabling problems, they are usually incomplete in some aspect.

Cooper and Kingston [14,15,16] propose a formal specification of the problem based on TTL, a timetabling specification language. A TTL instance consists of a time group, a set of resource groups, and a set of meetings. A time group defines the identification of the time slots available for meetings, followed by a specification of the way in which time slots are distributed over the days of the week. For specifying this distribution, a format is proposed, based on the utilisation of brackets (to enclose days), colons (meaning breaks), dots (for preferable time slots) and commas (for undesirable days). Resource groups can contain subgroups, which are subsets of the resource set defining functions that they may perform. A resource may be in any number of subgroups. In this specification language, meetings are collections of slots that are to be assigned elements of the various resource groups, under certain constraints. Only a basic set of constraints is defined in the language specification.

Cumming and Paechter propose a standard data format in a discussion paper presented at PATAT'95 conference [18], but not submitted formally to the conference or printed in the proceedings. Their proposal was highly criticised at the conference for lack of generality but generated a big discussion about the subject in which the difficulty of creating such a standard became evident. In their discussion paper [18], Cumming and Paechter propose principles and requirements to guide the creation of the standard. Their standard claims to represent complete and incomplete timetables and preferences. The components used are time slots (using a day.hh:mm representation format), events, staff and students, and rooms. They make no distinction between staff and students arguing that in some cases students can lecture classes. A list of keywords with different parameters is proposed as the standard. For example, offer.room with parameters event and room, represents that a given event must be assigned a room and that the specified room is an option. They also propose a cross convention (but not as part of the standard) that may be attached to any keyword and enables a Cartesian product between the arguments of that keyword (lists in this case). They also attempt to represent the soft constraints using cost functions. Moreover they conclude that timetable evaluation is likely to be the most difficult part to standardise. Some important omissions of this work are concerned with the availability of resources, split events, groups of resources, weeks and other type of periods, room types, definition of what is the problem to solve and how to represent the solution.

An interesting work related to standard data format for timetabling problems is included in the GATT timetabling system [13] by Hart (former Collingwood), Ross

and Corne. GATT ("Genetic Algorithm Time Tabler") uses a file format for describing timetabling problems. The format claims to be able to describe any GELTP ("General Exam/Lecture Timetabling Problem") and also non-educational timetabling problems. The file format is verbose and uses as main components: events, time slots, rooms, students and teachers. The format was essentially devised for exam timetabling problems. This way, some important omissions include weeks and other periods (useful for staff allocation and university course timetabling problems), room types, event sections (and section duration), continuity and load constraints (useful to achieve good quality schedules for teachers and students), student groups (essential in school timetabling) and teacher groups.

A more recent paper by Burke, Kingston and Pepper [10], proposes a different kind of standard for timetabling instances. They include a simple but incomplete description of the data types, keywords and syntax of the language and outline some further facilities to develop. Some concepts and constructs they use are similar to those found in the Z specification language [36]. The format includes as data types: classes, functions, sets, sequences, integers, floats, booleans, chars and strings. Classes include some attributes and functions and an inheritance mechanism is provided. A useful data type of this work are sets since many of the components of timetabling instances involve groups of resources (groups of students, groups of classes, etc.). In addition to the common set operators (member, union, intersection, subset, etc.), they also include operators like forall, exists, sum and prod. All these data types make the specification language close to a kind of programming language and enable the definition of constraints in logic programming language style.

A good extension of Burke, Kingston and Pepper's work could be the use of a constraint logic programming language as the specification language. Logic, associated with constraints performs very well in describing and solving timetabling problems. However, logic is not appropriate as an interchange format between computer specialist and school administrators. Therefore, we here propose a simpler and verbose language as our language for describing timetabling problems.

3 Sub-Problems of the Timetabling Problem

Timetabling can be viewed as a multi-dimensional assignment problem [11] in which students and teachers (or invigilators) are assigned to courses, exams, course sections or classes and events (individual meeting between teachers and students) are assigned to rooms and time slots. This multi-dimensionality indicates that we do not have a single problem designated by timetabling. We can have student assignment, teacher (or invigilator) assignment, room allocation and time allocation problems, all included in a given global timetabling problem. Figure 1 shows a generic timetabling system. The description of a given problem must be pre-processed in order to perform validity checks and decompose the original problem in its associated sub-problems. The sub-problems after being solved, using appropriate algorithms, enable the construction of the timetabling problem final solution.

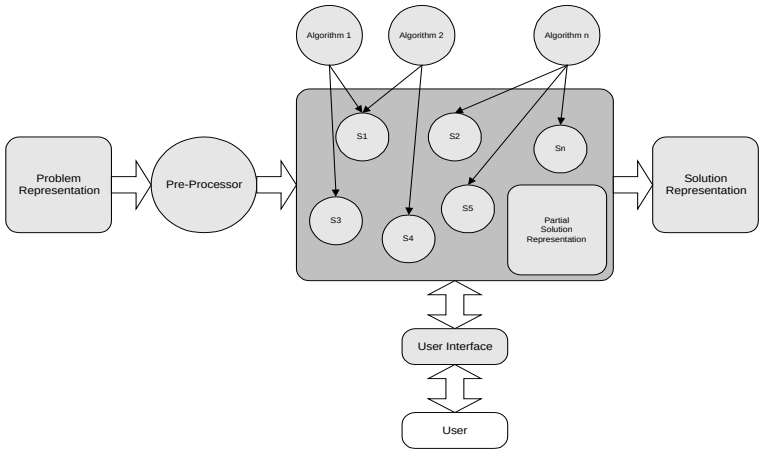

Fig. 1. A generic timetabling system.

Extending Carter's definition of the timetabling sub-problems [11], the problem analysis led us to the identification and definition of the following eight classes of inter-related timetabling sub-problems:

- **CTT - Class-Teacher Timetabling:** This is a common problem in most high schools and consists in allocating a time (or set of times) to each lesson of each module in the school (or university). The scheduling unit is a group of students (or class) that has a common program. It is considered that the assignment of teacher and classes to the events has already been made. Usually, either rooms are used as constraints in this problem or room allocation is performed simultaneously with class-teacher timetabling.
- **CT - Course Timetabling:** Scheduling of all the lectures of a set of university degree modules, minimising the overlaps of lectures having common students and avoiding teachers double booking. In this case, students are considered individually and are not assumed to belong to some group as in school timetabling. Teachers are usually already assigned (although some flexibility may be included in this assignment). Sometimes, students are not assigned to the event sections before the course timetabling scheduling. Although they are usually already assigned to the events, in some universities this can also be untrue.
- **ET - Examination Timetabling:** Scheduling (in time) of the exams of a set of university courses avoiding overlapping exams having common students and spreading the exams for the students as much as possible. Room assignment and invigilator assignment can be done prior or after the exam timetabling phase.
- **SD – Section Definition:** This problem occurs within every timetabling problem. It consists in, for each event, to define the number of sections offered. Each section is simply a different occurrence of the same event (for example, the same lecture given to a different group of students). A similar problem occurs in examination timetabling if we consider split examinations (in time).
- **SS - Student Scheduling:** This problem occurs when modules are taught in multiple sections. Once students have selected their modules, they must be assigned to module sections, trying to provide schedules (for students) without conflicts and balancing section sizes.
- **SA - Staff Allocation (Teacher Assignment):** Assigning teachers to different modules, trying to respect their preferences, including preferred subjects, balance between lecture hours in different periods of the year (semesters or trimesters) and other side constraints. Sometimes this is achieved in two or three consecutive separated steps. The first step

consists in selecting teachers responsible for each one of the modules (although in traditional public universities this problem is solved doing only small variations from year to year). In the second step, the teachers that will lecture each one of the modules are selected (along with their own workloads). A third step consists in assigning individual sections to teachers. Sometimes this last phase remains opened (or at least very flexible) until the timetabling generation phase.

- **IA - Invigilator Assignment:** This is an usual problem which is associated to examination scheduling. Each exam needs one or more invigilators. The number of invigilators is related to the number of students having the examination and to the number of rooms needed.
- **RA - Room Assignment:** Usually all real timetabling problems have a room assignment phase. Events must be assigned to specific rooms (or sets of rooms) satisfying the size and type required for the event. Problems with split events, shared rooms and distances between rooms for events may arise.

Other sub-problems could be included in this classification. However we consider those other problems as unusual or unimportant compared with the described ones. For example, prior to examination timetabling, some institutions may aggregate small exams into sets that will be scheduled at the same time using the same room and supervisors. Other institutions use modules with different workloads throughout the year. This way, the timetables are different every week and the problem becomes a kind of mix between timetabling and job-shop scheduling in which module workloads are defined for each week and its lessons are scheduled in each week. However, we believe that the sub-problems we took into consideration include the timetabling problems faced by most of the existing schools and universities around the world.

To describe a specific timetabling problem, it is necessary to know in detail, which ones of the sub-problems will be solved and in which order shall they be solved. Some flexibility is allowed in this ordering by some universities or schools, while others obey strict and rigid rules.

4 Requirements for the Proposed Language

Many formalisms for timetabling application descriptions have been designed including concerns for minimising data storage space or to facilitate fast data processing. Our proposed formalism tries to compromise between generality and simplicity. It is sufficiently general to allow representation of the most common variations of timetabling problems, including school, university and examination-timetabling keeping also a very simple syntax. After a series of interviews with timetabling experts, we arrive to the following main requirements associated with the needed language:

- It should be independent of implementation details and timetabling strategies;
- It should be easy to extend, including new concepts and constraints;
- Existing benchmark problems should be easily translated to the new formalism;
- The new formalism should be compatible with most timetabling systems and easily readable by the human user;
- Information not directly concerned with the scheduling problem should not be included in the formalism. Therefore, common information regarding school administration, like student names and addresses, is omitted;
- The formalism should be general enough to enable the representation of school timetabling, university timetabling and exam timetabling problems

- It should be suitable for representing complete problems and simple related sub-problems (like staff or invigilator assignment, section definition and room assignment);
- It should include a clear definition of all the constraints (both hard and soft) associated with the problems;
- There should be possible to represent both incomplete and complete solutions;
- A timetabling quality evaluation function should be easy to include, enabling to evaluate directly any proposed solution;
- It should be as concise as possible;
- It should be robust, enabling simple data validation and override of common errors.

5 Language for Representing Timetabling Problems

Based on the identification of the eight timetabling sub-problems presented in Section 3 and on the requirements presented in the previous section, we propose a new language called UniLang for representing timetabling problems. UniLang intends to be a standard suitable as specification language for any timetabling system. It enables a clear and natural representation of data, constraints, quality measures and solutions for different timetabling (as well as related) problems, such as school timetabling, university timetabling and examination scheduling.

5.1 Components

The model behind our proposal includes the definition of different *time periods* composed of a set of *weeks* (or other *time period*) with equal (or similar) schedules. Each *week* is composed of a set of *time slots* located at a given *time* of a given *day*.

We prefer to separate resources definition into three main classes: *students*, *teachers* and *rooms*. Although some similarities exist between the three (for example, all the resources have availability constraints), the differences are evident in any timetabling problem. For example, *rooms* can hold (in some timetabling problem) several *events* at the same *time*, have capacity constraints, types and distances between them. *Students* and *teachers* can be put together in groups (although with different meanings) and have workload constraints. *Teachers* can have maximum and minimum *event* assignment constraints and ability constraints (stating if they are capable or willing to lecture/supervise a given subject/exam). *Students* (and *student groups*) have also associated spreading and continuity constraints. As a consequence, we have considered the following components: *Periods*, *Time Slots*, Resources (*Rooms*, *Teachers* and *Students*) and *Events*.

An *event* is a meeting between a *teacher* (or a set of *teachers*), a set of *students* (or *student groups*) that takes place in a *room* (or in a set of *rooms*) in a given *time slot* (or set of *time slots*). We do not consider (explicitly) other resources like equipment, because these situations are very rare in timetabling problems.

Our model also includes the specification of both data and constraints in a common format. Since sometimes the distinction between data and constraints is not completely clear, this seems an adequate approach. Some of the types of constraints included have two different versions: A strong version in which the constraint must be respected (hard constraint) and a weak version that should be respected only if possible (soft constraint). The considered set of soft constraints enables the definition of a timetabling quality function. Each one of the soft constraints included has

associated a numerical *preference* that really is the cost of not satisfying that constraint.

A solution (partial or global) to a timetabling problem consists in, for each one of the *events* considered, a set of *slots*, a set of *rooms*, a set of *teachers* and eventually, a set of *students* (or *student groups*). Although in most of the timetabling problems found in the literature, *students* and *teachers* are already assigned to the *events* (or *event sections*), our model intends to be general enough to solve also those assignment sub-problems, (i.e. *teacher assignment*, *invigilator assignment* and *student scheduling*).

To make the formalism still more robust, and following Hart's idea in the Gatt system [13], a list of synonyms (Table 1) may be included for each one of the concepts (keywords) used.

Table 1: Example of a brief list of synonyms for the keywords used in the language.

```
[Default | All | Every | Any], [Year | Problem | File], [Schedule | Timetable],
[Event | Module | Lecture | Lesson | Exam | Examination | Tutorial],
[Teacher | Teachers | Invigilator | Supervisor | Lecturer], [Student | Students],
[Day | Days], [Time, | Times | Hour | Hours | Minute | Minutes], [Slot | Slots | Time Slot],
[Place | Places | Room | Rooms], [Preference | Weight | Priority | Penalty],
[Consecutive | Continuos], [Simultaneously | Concurrently | Together | At the same time],
[Teaches | Invigilates | Supervises], [Hold | Holds | Have capacity | Has capacity | Capacity],
[Specify | Specifies | Require | Requires | Need | Needs], [Last | Lasts], [Duration | Length],
[Contains | Comprises | Has | Have], [Is | Are],
[At least | No less than | No fewer than], [At most | No more than], [Exactly | Precisely],
[Group_teachers | Area | Department], [Group_students | Class | Student_type],
[Room_type | Room_Group | Buildings], [Double_bookings | Clashes | Conflicts]
```

The use of these synonyms makes the description of input data files more easily readable by human users. Also, users can build their own list of synonyms in a separate file and, with the help of a simple pre-processor, that file can be directly converted into our proposed specification language.

5.2 Time Representation

Each file contains a problem description concerning a specific *period of time*: A school *year*. Therefore, the file needs the identification of that specific *year*:

```
this is year <NAME>
```

Each *year* can be divided into smaller *time periods* (semesters, trimesters, exam periods, etc). UniLang enables defining the number of *periods* used (for the problem) and the names of those *periods*. Each *period* may have a *starting date*, *starting and ending weekdays* and it is composed of a set of *weeks*. The concept of *week* is not that of a typical *week*. Here we are interested in a concept of *week* as a *period of time* to which the same schedule has been assigned. So, for example, in an examination timetabling problem, a '*week*' may last 15 or 20 *days*, while in a typical university or school timetabling problem, a *week* lasts 5 or 6 *days*.

```
periods are {<PE>}
period {<PE>} contains weeks {<N>}
period <PE> begins on date <DATE>
period <PE> begins|ends on day <D>
```

We then assume that each *week* is composed of a set of consecutive *days* and that each *day* is composed of a set of disjoint and consecutive *time slots*. Each *slot* is located in a given *day* and begins at a given *time*.

```
slots are {<S>}
days are {<D>}
times are {<T>}
slot <S> is on day <D> at time <T>
```

If the *slots*, *days* and *times* are explicitly defined, then the previous sentence can be simplified. For example, if <*S*> is a *slot*, <*D*> is a *day* and <*T*> is a *time* then "*slot <S> is on day <D> at time <T>*" can be described in one of the following ways:

```
<S> is on <D> at <T>
<S> <D> <T>
```

This simplification applies to all keywords that come before any other component in a sentence (like *teacher*, *student*, *room*, *teacher_group*, *event*, etc.).

Some *slots* cannot be used for timetabling *events*. Examples are *slots* on Sundays or Saturdays and *slots* at night. Unilang enables two ways of stating this impossibility. The first one is simply not defining the impossible *slots*. The second is based in defining the *slots* and stating explicitly the impossibility of the allocation. This enables coherent time difference measurement between *slots* that may be needed for some timetabling applications.

```
slots {<S>} are unusable
```

We can define *time periods* (like mornings, afternoons, lunch times, etc.) using the concept of *time_period*. Each *time period* is composed of a set of *slots* in a *week*. Unilang includes the keyword '*default*' that enables assigning values to all elements of a class.

```
time_periods are {<TP>}
time_period {<TP>}|default contains slots {<S>}
```

Using the keyword '*default*' the constraints are applied to all the elements belonging to the given class. This can also be applied to any constraint that deals with *time slots*, *rooms*, *teachers*, *students* and *groups* (of *teachers* or *students*).

To each *slot*, a capacity in terms of both *events* and *seats* can be assigned. These capacities are important to enable the definition of timetabling problems that do not have associated *room allocation* problems. Otherwise, if *room allocation* is a part of the complete problem to solve, *slot capacities* are not needed to be stated explicitly.

```
slots {<S>}|default have capacity <N> seats|events
```

To state that the problem of time allocation for the described *events* has to be solved, the following syntax is provided.

```
solve class-teacher timetabling
solve course timetabling
solve examination timetabling
```

The three last lines express how to ask for solutions of the three broad classes of timetabling problems. Knowing the timetabling problems to solve, a given timetabling system may then select the appropriate constraints and quality measures for those problems.

5.3 Space Representation

Our space representation is based on the idea that each *event* usually needs one *room* but may, in some cases, be hosted by more than one *room* or even do not need a *room*. Our formalism enables the definition of the *rooms* available in the following way:

```
rooms are {<R>}
```

Each *room* can hold a given number of *students* at a *time*. In a similar way, each *room* can hold only a given maximum of *events* (usually one) at a *time*. If *preference* is omitted, this becomes a hard constraint. If *preference* is included then the constraint can be violated with a cost *P*.

```
room {<R>}|default holds <N> students|events [preference <P>]
```

The number of *students* and *events* that a *room* can host simultaneously can be different throughout the *week slots*.

```
room {<R>}|default holds <N> students|events in slots{<S>}
[preference<P>]
```

Usually, *rooms* are not allowed to have *double booking* and may be *unavailable* or preferably usable during some *time slots*.

```
room {<R>}|default cannot have double_bookings [preference <P>]
room <R>|default specify|excludes in slots {<S>} [preference <P>]
```

Our formalism provides also *room types*. Each *room* has a given *room_type* or set of *room_types*:

```
Room_types are {<RT>}
room_type {<RT>}|default has rooms {<R>}|default [preference <P>]
```

We include also the concept of distance between *rooms*. A *room* can be connected (if the preference is omitted) to another *room* or can be close to another *room*, with a given proximity measure:

```
room {<R>} close to room {<R>} [preference <P>]
```

To state that the problem includes the assignment of *rooms* to *events*, the language proposes the following syntax:

```
solve room assignment
```

5.4 Event Description

The concept of *event* is crucial in our specification. An *event* can be a lecture, exam, tutorial, lunch, etc. It has a given duration (in terms of *time slots*), may or may not need space (*rooms*) and resources like *teachers* (or *invigilators*) and *students* (or *student groups*).

The first step is to define the *events* (*event* identifiers) and, eventually, some groups of *events* (like degrees, degree years, etc.).

```
events are {<E>}[in period <PE>]
groups_events are {<GE>}
```

Events may have default duration but may also be given a different individual duration. The *students* that may/will attend the *event* have also to be defined

```
event {<E>}|default lasts <N> slots
event {<E>} has students {<ST>} [preference <P>]
```

Besides the number of *students* expected, an *event* can have some (anonymous) external extra *students*. The total number of *students* of the *event* can also be defined directly. This can be useful in cases where *students* are not going to be considered individually. Some *events* may also require a given number of (anonymous) *teachers* or *rooms*. This can be useful if we are not concerned with the *teacher* or *room allocation* problems.

```
event {<E>} has <N> students|extra_students
events {<E>}|default requires <N> teachers [preference <P>]
events {<E>}|default requires <N> rooms [preference <P>]
```

An *event* usually needs a given type of *room* and *events* may belong to groups:

```
events {<E>}|default requires room_type {<RT>} [preference <P>]
group_events {<GE>} has events {<E>} [preference <P>]
```

Events can be split into a number of different *parts* of different duration that will occur in different *days* of the *week*. A typical constraint concerning *event parts* is that they should not be assigned on the same *day*. *Events* can also be divided into multiple *sections*, meaning that different *teachers* can repeat them during the *week* to different

students. For example, an exam can have two *parts* (a theoretical and a practical part) and three *sections* (one for class A, one for class B and another for class C). A module is usually divided into lectures (*event parts*) that are taught in different days of the week and may have multiple *sections* (taught to different *students*).

```
event {<E>}|default has <N> event_parts of duration {<N>}
event {<E>}|default has <N> event_sections
event {<E>}|default minimum/maximum <N> students
```

If *event sections* are not defined and the *section definition* problem must be solved, then, we have:

```
solve section definition
```

5.5 Classes and Students

The number and names (identifiers) of *students* and *student groups* can be declared just in the same way as the names of *slots*, *rooms* and *events*:

```
students are {<ST>}
group_students are {<GST>}
```

The *students* can be grouped into *student groups* (with common, or similar, schedule preferences or degree plans):

```
group_students {<GST>} have students {<ST>}|default [preference <P>]
group_students {<GST>} has <N> students
```

A *student* can also be a *teacher*. This is the case of postgraduate *students* that also are lecturers to their undergraduate colleagues.

```
student <ST> is teacher <TE>
```

Students or *student groups* can be unavailable in some *time slots* throughout the *week*. This can be a hard or a soft constraint:

```
student|group_students {<ST>|<GST>}|default specify|excludes slots
{<S>}|default [preference<P>]
```

Students and *student groups* may attend *events*:

```
event {<E>}|default has students|group_students {<ST>|<GST>}|default
[preference <P>]
```

A *student* or *student group*, besides being registered in an *event* can also be registered in one of its *sections*. *Preferences* can also be used for allocating *students* and *student groups* to *events*:

```
event_section {<ES>} has students {<ST>} [preference <P>]
event_section {<ES>} has group_students {<GST>} [preference <P>]
```

To prevent that *students* (or *student groups*) have *double booking*, we can declare:

```
student|group_student {<ST>|<GST>}|default cannot have
double_bookings [preference <P>]
```

Those declarations can be seen either as soft constraints (with a given violating cost *<P>*) or hard constraints (if the keyword *preference* is not used). If the problem of allocating students to *event sections* is considered, then this must be stated using the following:

```
solve student scheduling
```

5.6 Teachers and Invigilators

In the proposed model, *teachers* can be seen both as lecturers and invigilators (and the keyword *teaches*, can also signify *invigilates*). *Teachers* can be grouped in areas or *groups of teachers*. This leads to the definition of departments and scientific areas.

```
teachers are {<TE>}
```

```
group_teachers are {<GTE>}
group_teachers {<GTE>}|default has teachers {<TE>}|default
```

Each *teacher* can be previously assigned to teach a given number of *events* (pre-allocations). The information of each *teacher* is also concerned with his capability of teaching a given *event*.

```
teacher {<TE>} teaches|cannot_teach events {<E>} [preference <P>]
```

In order to be able to describe *staff allocation* problems and *invigilator assignment* problems, *maximum* and *minimum* number of *events* (or *times*) for *teachers* are needed.

```
teacher {<TE>}|default maximum|minimum<N> events|times[in period<PE>]
```

Teachers or *groups of teachers* may also have timetable *preferences*. So a *teacher* (or *group*) can exclude (or avoid with some *preference*) specific *time slots*:

```
teacher|group_teachers {<TE>|<GTE>}|default specify|excludes slots
{<S>}|default [preference <P>]
```

Usually *teachers* cannot have *double booking* and sometimes, depending on the meaning of *teacher groups*, these *groups* cannot have *double booking* too.

```
teacher|group_teacher {<TE>|<GTE>}|default cannot have
double_bookings [preference <P>]
```

If the problem includes assigning *teachers* to *events* (i.e. *staff allocation*, or *invigilator assignment*) then, the following line should be included:

```
solve teachers assignment
solve invigilator assignment
```

The order in which different sub-problems will be solved does not concern this language. Unilang defines the complete problem and states the timetabling sub-problems to be solved. It is up to the solver, by analysing the complete problem description, to choose the order by which different sub-problems will be solved.

5.7 Time and Space Preferences

We have two different kinds of *time preferences*: regarding *time slots* and regarding *time periods* (mornings, afternoons, etc.). These preferences include pre-allocations (*specify*) and exclusion or avoidance (*excludes*), with *preferences*.

```
event {<E>}|default specify|excludes slots {<S>} [preference <P>]
event {<E>}|default specify|excludes time_period{<TP>}[preference<P>]
```

Preferences associated with space are just like *time slot preferences*. There are two types of those: *room preferences* and *room_type preferences*.

```
event {<E>}|default specify|excludes rooms {<R>} [preference <P>]
event {<E>}|default specify|excludes room_types{<RT>} [preference<P>]
```

5.8 Workload, Spreading and Ordering Constraints

The workload and spreading constraints are concerned primarily with the utilisation of *teachers* and *students* throughout the scheduling period. *Students* and st*udent groups* may have workload constraints stating that they cannot have more than a given number of *events* or a given number of *time slots* of work in a row (*consecutive*) or in a given *day*. Usually, these are *at most* constraints but the *exactly* and *at least* constraints can also be used in some, less frequent, problems. These constraints can be also applied to *teachers*:

```
students|student_group {<ST>|<GST>}|default have
exactly|atleast|atmost <N> [consecutive] events|times in a day
[preference <P>]
```

```
teacher {<TE>}|default have exactly|atleast|atmost <N> [consecutive]
events|times in a day [preference <P>]
```

Spreading constraints are typical in examination scheduling problems. Through them it is achieved that *students* (or *student groups*) have sufficient time intervals between *events*. Usually these are *at least* constraints.

```
students|student_group {ST|GST}|default have exactly|atleast|atmost
<N> times|days between events [preference <P>]
```

Sometimes, spreading constraints state that *students* have *at most* a given number of *events* (or occupied *times*) in each number of *days* (or *times*):

```
students|student_group {ST|GST}|default have exactly|atleast|atmost
<N> events|times in each <N> times|days [preference <P>]
```

The ordering constraints are concerned with the sequential order of some *events* in the scheduling period. An *event* can be scheduled *exactly*, *at least* or *at most*, a given number of *times* (or *days*) before another *event*. Another usual type of constraint sates that there are *exactly*, *at least* or *at most* a given number of *times* or *days* of interval between two *events* (without any concern of which *event* comes first).

```
events {<E>} exactly|atleast|atmost <N> times|days before|interval
events {<E>} [preference <P>]
```

Another typical constraint states that a given number of *events* occurs or cannot occur *simultaneously* or *on the same day*.

```
events {<E>} are [not] simultaneously|on_the_same_day [preference<P>]
```

5.9 Override and Missing Mechanisms

Two useful mechanisms in our language are the override and the missing mechanisms. The override mechanism enables the redefinition of some concept in a stronger way, overriding the previous definition. The missing mechanism enables that some concepts that are not formally defined in our problem description may be deduced from the problem description. These mechanisms are fully employed in our translator from this specification language to a constraint logic program.

5.10 Evaluation Function

The evaluation function for a given timetabling problem is implicitly defined through the definition of the problem data and constraints. The hard constraints must be respected. The soft constraints (in which the keyword *preference* appears) should be respected to achieve a good quality solution. For each of the soft constraints violated, the *preference* value (<*P*>) is added to the total penalty associated with that specific solution. The smaller this value is, the better the final solution is.

5.11 Representation of the Final Solution

The final solution of any timetabling problem always include for each *event part*, a set of *time slots* (when the *event parts* happen), a set of *rooms* (where the *events* take place), a set of *teachers* (or *teacher groups*) that will lecture (supervise or participate) the *event* and a set of *students* (or *student groups*) that will attend the *event*. In Unilang this is specified using the following notation:

```
event_part <EP>|event_section <ES>|event <E> is in slots {<S>}
event_part <EP>|event_section <ES>|event <E> is in rooms {<R>}
event <E>|event_section <ES> is taught by {<TE|GTE>}
```

```
event <E>|event_section <ES> has students {<ST|GST>}
```

For each of the *events*, event sections (if they exist) or event parts (if they are specified), a set of *time slots* and a set of *rooms* may be specified. The solution is completed with a set of *teachers* (or *teacher groups*) and a set of *students* (or *student groups*) for each of the *events* (or *event sections* if they are specified). It is assumed that each *event section* has the same *teachers* and *students* for each of its *parts*. We have also developed a simple evaluator that takes as inputs a simple problem (specified using our language) and a solution for the problem and calculates (using this data) a numeric value denoting the solution quality.

6 Translating and Solving Timetabling Problems

To enable the resolution of any kind of timetabling problem, which is represented using the proposed language, we have implemented a simple translator that converts this representation to a Constraint Logic Program [28,43] in ECLiPse language [22]. To solve the problem, the user shall then specify the labelling strategy. The solution is then automatically obtained. Our constraint logic programming approach uses, internally, finite domain variables for the starting times of events, finite domain variable for single room or single teacher assignment problems and set variables [24] for rooms, invigilators/teachers or students in multi assignment problems.

After the translation of the problem definition, the user may specify the ordering in which the different types of variables are labelled [38]. For example, in a complete examination timetabling problem, we may have finite domain variables for exams starting times, set variables for rooms and set variables for invigilators. The user may specify that the rooms would be labelled first, then times and lastly invigilators. Another possibility is, for each event, to assign first, simultaneously, time and rooms and then, at the end, perform the easier invigilator assignment. All combinations of the different types of variables are available to the user. For more details about our methods for solving timetabling problems which are described using this language and using different constraint logic programming approaches, see [37,38,39].

7 Conclusions and Future Work

This is, to our knowledge, the first attempt to use the identification of the subclasses of the timetabling problem to guide a possible standardisation of the data and constraints that define any complete timetabling problem. We do not ignore that there are lots of open issues in this “standard-like” definition and some characteristics are still missing in our language to enable the complete representation of specific problems. Representation of soft constraints is not complete, in the sense that any user can have a specific individual quality measure. However if it is specific and individual, then it should not be included in the standard. Some people may argue that the definition of the constraints should also appear explicitly (and not implied by its meaning). The problem is that doing so, we lose the simplicity of this verbose format to enter in a more complicated logic or functional format. We once more emphasize that, since the beginning, one of our requirements is that administration staff may use the proposed format to describe, by their own, their timetabling problems.

To enable the easy use of our specification language, we already have implemented a simple translator (including the override and missing mechanisms) from this definition language to a constraint logic programming language. The translator was tested against complete examination together with some class-teacher timetabling problems (with different sub-problems to solve). Several real problems were represented in the language, successfully translated and, after selecting appropriate labelling strategies, they were successfully solved.

We foresee the future work as related with developing a library for complete educational timetabling problem descriptions. Based on Unilang we are building such a library of timetabling problems, including the eight timetabling sub-problems presented. Moreover, complete timetabling problems, including several frequent combinations of the eight previous ones are also being addressed. We are analysing both real timetabling problems and randomly generated ones. For each problem, quality measures and the already known best solution are included. We believe that all this future work will make it possible easy and fast comparison between the solutions of typical timetabling problems, achieved by different researchers, using different algorithms.

Acknowledgement

This work was partially supported by a research grant PRAXIS XXI, BD/5663/95 of the Portuguese Foundation for Science and Technology.

References

1. Abramson, D. and Dang, H., School Timetables: A Case Study Using Simulated Annealing, Applied SA, Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, pp.104-124, 1993
2. Appleby J.; Blake D. and Newman E.., Techniques for Producing School Timetables on a Computer and Their Application to other Scheduling Problems, The Computer Journal, Vol. 3, pp.237-245,1960
3. Bardadym, V. Computer-Aided Lessons Timetables Construction. A Survey, USIM – Management Systems and Computers, Vol. 8, pp. 119-126, 1991
4. Barham, A. and Westwood, J. A Simple Heuristic to Facilitate Course Timetabling, Journal of the Operational Research, Vol. 29(11), pp. 1055-1060, 1978
5. Burke and Ross (Eds.), The Practice and Theory of Automated Timetabling, LNCS, Vol. 1153, 1996
6. Burke E. K.; Elliman D. G and Weare R. F, A Genetic Algorithm for University Timetabling, in proceedings of the AISB workshop on Evolutionary Computing, University of Leeds, UK, April 1994
7. Burke E.; Elliman D., Ford P., Weare R., Examination Timetabling in British Universities - A Survey, 1st Int.Conf. Practice and Theory of Automated Timetabling PATAT'95, Edinburgh, pp.423-434, 1995
8. Burke, E.; Carter M. (Eds.), Practice and Theory of Automated Timetabling II, LNCS, Vol. 1408, 1998
9. Burke, E., Jackson, K., Kingston, J. and Weare, R., Automated Timetabling: The State of the Art,
10. Burke, E., Kingston, J, Pepper,P. A Standard Data Format for Timetabling Instances, in [8], pp.213-222, 1998
11. Carter, M. and Laporte, G., Recent Developments in Practical Examination Timetabling, in [8], pp. 3-21, 1996
12. Carter, M., A Survey of Practical Applications of Examination Timetabling Algorithms, Operations Research, Vol. 34(2), pp. 193-202, 1986
13. Collingwood, E., Ross, P. and Corne, D. A Guide to GATT, University of Edinburgh, 1996
14. Cooper, T. and Kingston, J., A Program for Constructing High School Timetables, Proc. 1st Int. Conf. on the Practice and Theory of Automated Timetabling, Napier University, Edinburgh, 1995
15. Cooper, T. and Kingston, J., The Complexity of Timetable Construction Problems, 1996

16. Cooper,T.; Kingston,J., The Solution of Real Instances of the Timetabling Problems, The Computer Journal, Vol. 36, pp. 645-653, 1993
17. Corne, D. and Ross, P. and Fang, H., Fast Practical Evolutionary Timetabling, Proceedings of the AISB Workshop on Evolutionary Computing, Springer-Verlag, 1994
18. Cumming, A. and Paechter, B., Seminar: Standard Timetabling Data Format, International Conference on the Practice and Theory of Automated Timetabling PATAT'95, Edinburgh, UK, 1995
19. de Werra, D., An Introduction to Timetabling, Eur. Journal of Oper. Res., Vol. 19, pp.151-162, 1985
20. de Werra, D., Construction of School Timetables by Flow Methods, INFOR – Canadian Journal of Operations Research and Information Processing, Vol. 9, pp. 12-22, 1971
21. Dempster M., Two Algorithms for the Time-Table Problem, Welsh D.J.A. (ed.) Combinatorial Mathematics and Its Applications, Academic Press, pp. 63-85, 1971
22. Eclipse User Manual, Aggoun et al., ECRC GmbH, 1992, Int.l Computers Limited and IC-Parc, 1998
23. Fahrion, R. and Dollanski G. Construction of University Faculty Timetables Using Logic Programming Techniques, Discrete Applied Mathematics, Vol. 35, pp. 221-236, 1992
24. Gervet, Carmen, Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language, Constraints, An International Journal, Vol. 1, pp. 191-246, 1997
25. Gotlieb,C. The Construction of Class-Teacher Time-Tables, Proc.IFIP Cong, Munchen, pp.73-77,1963
26. Gudes E.; Kuflik T. and Meisels A. On Resource Allocation by an Expert System, Engineering Applications of Artificial Intelligence, Vol. 3, pp. 101-109, 1990
27. Gueret, Christelle et al, Building University Timetables Using Constraint Logic Programming, Proc. of 1st Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT '95), pp.393-408, 1995
28. Hentenryck,P. Constraint Satisfaction in Logic Programming, LP Series, MIT Press, Cambridge,1989
29. Hertz, A., Finding a Feasible Course Schedule using Tabu Search, Discrete Applied Maths, Vol.35, pp.225-270, 1992
30. Junginger, W., Timetabling in Germany – a Survey, Interfaces, Vol. 16, pp. 66-74, 1986
31. Kang, L. and White, G. A Logic Approach to the Resolution of Constraints in Timetabling, European Journal of Operational Research, Vol.61, pp.306, 317, 1992
32. Kingston, J, A Bibliography of Timetabling Papers, URL ftp://ftp.cs.su.oz.au/jeff/timetabling/
33. Kingston,J.; Bardadym,V. ; Carter, M., Bibliography on the Practice and Theory of Automated Timetabling, in ftp://ftp.cs.usyd.edu.au/jeff/timetabling/timetabling.bib.gz,, Univ. Sidney, 1995
34. Kovacic, M. Timetable Construction with a Markovian Neural Network, European Journal of Operational Research, Vol. 69, 1993
35. Lawrie, N., An Integer Programming Model of a School Timetabling Problem, The Computer Journal, Vol. 12, pp. 307-316, 1969
36. Potter, B. et al, An Introduction to the Formal Specification Language Z, Prentice Hall, 1991
37. Reis, L. P. and Oliveira, E., A Constraint Logic Programming Approach to Examination Scheduling, AICS'99, Artificial Intelligence and Cognitive Science Conference, Cork, Ireland, September 1999
38. Reis,L.P. and Oliveira,E., Constraint Logic Programming using Set Variables for Solving Timetabling Problems, INAP'99, 12th Intern. Conf. on the Applications of Prolog, Tokyo, Japan, September 1999
39. Reis, L. P., Teixeira, P. and Oliveira, E., Examination Timetabling using Constraint Logic Programming, ECP'99, 5th European Conference on Planning, Durham, U.K., September 1999
40. Schaerf, A., A Survey of Automated Timetabling, TR CS-R9567, CWI–Cent. Wiskunde en Informatica, 1995
41. Schmidt, G.; Strohlein, T., Timetable Construction – An Annotated Bibliography, The Computer Journal, Vol.23 (4), pp. 307-316, 1979
42. Stamatopoulos, P. et al, Nearly Optimum Timetable Construction Through CLP and Intelligent Search, International Journal on Artificial Intelligence Tools, Vol. 7, No. 4, pp. 415-442, 1998
43. Tsang, E., Foundations of Constraint Satisfaction, Academic Press, Inc, 1993
44. Verbraeck, A. A Decision Support System for Timetable Construction, International Conference on Expert Planning Systems, Brighton, England, pp. 207-211, 1990
45. Welsh, D. and Powell M., An Upper Bound for the Chromatic Number of a Graph and its Applications to Timetabling Problems, Computer Journal, Vol. 10, pp. 85-86, 196
46. Wren, A. Scheduling, Timetabling and Rostering – A Special Relationship, in Burke and Ross (eds.), The Practice and Theory of Automated Timetabling, Springer LNCS, Vol.1153, pp.46-76, 1996