A School Timetable Description Language

Samir Ribić, *Member, IEEE*

***Abstract* — Domain specific languages are used in situations when general-purpose languages or classical user interface are not best suitable for describing the problem and solution. During automatic school timetable generation, there are different requirements that need to be satisfied and they might clutter the user interface. There is a proposal with implementation details of the domain specific language designed for explaining many different constraints that appear in timetable description: the language syntax, some regular sentences, required data structures for implementation and example of generated equations.**
***Keywords* — Timetabling, domain specific languages.**

I. Introduction

EVERY school, no matter elementary or university, needs to generate a time table, at least once a year. Preparing school timetable is a real world problem, and it is difficult in several ways, due to complexity of the problem itself, expressiveness in constraint definition, modeling and evaluation. Here we will look on several ways of describing timetable constraint requirements and propose one domain specific language, which defines properties of the desired timetable with implementation details, like required data structures used to keep the data after parsing.

Construction of timetable in schools is finding good combinations of four important resources: time slots, teachers, rooms and groups of students (classes). The timetable usually contains some constraints about room, teacher and class availability and preferences. The most obvious one is that teacher can not be at two places in a same time, or that every lesson must be assigned to exactly one time slot. There are several approaches in time table generation: do it in traditional way on paper; use a tool for manual timetable generation which will warn you about unsatisfied constraints; solve the problem completely automatically or use an automatically generated solution and then manually modify it

A manual approach is quite time consuming, and requires high level of concentration to the job. Even if we have good starting point (for example, timetable from a previous academic year), seemingly simple change in schedule requires many cascade changes. Say that teacher of Physics asked to move his lesson from 10:00 to 12:00. However, as his students are already assigned at 12:00 to attend a Geography course, we need to move the Geography course as well, which might require pushing English course, etc. However, manual timetabling has one advantage over automatically generated one: the social component, because teachers can interact with timetable generating person about requirements.

Samir Ribić, Elektrotehnički fakultet Univerziteta u Sarajevu, Zmaja od Bosne bb, Sarajevo, Bosnia and Herzegovina

II. Problem Definition

During domain analysis of this problem, we can remark than an automatic creation of school timetables has two aspects. First problem is in complexity of the solution and searching algorithm for the solution. This is a combinatorial problem with a large number of variables. Total number of possible timetables is 2^{tsrc}, where t is number of teachers, s is number of time slots, r is number of rooms and c is number of classes or other student groups. Only small percent of them are feasible timetables, and some of them can be considered as good ones. The problem is partially solvable using a variety of heuristic and optimization methods, integer linear programming, taboo search, genetic algorithms etc.

Much less explored problem is in defining requirements of the timetable. This question is related not only to automatic time table solving, but also for usage in software which only check constraints, or for documentation of manual timetabling.

Sometimes it is enough to enter a matrix with the names of teachers and classes in the matrix header, filled with weekly number of hours that the teacher teaches in the particular class. This can be done using simple graphical user interface like on Figure 1.

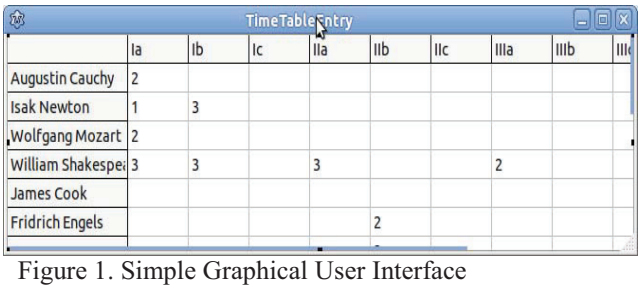

Figure 1. Simple Graphical User Interface

However, additional requests for special cases might be very different. Depending on situation, the classrooms may be assigned to the particular student group, course or not assigned at all. Some lessons require hint where it should occur, but for some courses we can use default values. The classes can be divided into smaller groups, or merged into common lessons. Some rooms can be unavailable at a particular time, as well as teachers and classes. In addition, some teachers or classes may prefer

978-1-4673-2984-2/12/$31.00 ©2012 IEEE

some time slots over other ones, sometimes pauses in time tables are desirable, sometimes not etc.

If we want to satisfy all the additional requirements using this grid, it would be necessary to open separate window, (for example by double click inside grid) where we can set up these requirements for the lesson. So to support different kinds of constraints using graphical interface, it will become so cluttered with many options, which might be not much used.

Many software applications in addition to graphical interface (for common usage) include scripting and domain specific languages for less common options. Domain specific languages are specialized for one group of problems. They typically solve its domain very well, but they are useless outside the domain. Most known examples are SQL, HTML, XML, TeX etc.

III. Current Domain Specific Languages For Combinatorial and Schedulnig Problems

Instead of the GUI, an alternative description of the problem may be the input file. There are standardized formats for describing the schedule requirements, for example the format used on time table generation competitions Also there are several domain specific languages specialized for combinatorial optimizations and scheduling. Essence [1] describes general optimisation problems using keywords “given”, “where”, “letting”, “find”, “maximising”, “minimising” and “such that”. ASPEN [2] is specialised for flight scheduling and describes problem in a form of activities, resources, constraints and timelines. Zinc [3] defines constraints in special sections and has built in support for set datatypes. OPL [4] is designed to solve wide class of optimization problems using non-procedural, but C-like syntax. Programs in TEMPLE [5], resemble C++ syntax and define intervals, links between intervals, derived properties, derived constraints, derived curves and initial solution. The domain specific languages intended for combinatorial optimizations do their job well. However, their usage requires good mathematical knowledge about underlying models or they resemble general purpose programming languages and therefore they are more suitable to programmers.

Even specialized file formats for time table description are defined for easier parsing, not to be easy to be understood, and they are not expandable. We need to use handbook of the file format during the entry, and this process is quite error prone. We can see it if we look at the input file used for timetabling competition found at [6]. Courses, rooms and curricula are there defined in special sections. Each description of course, room and curricula consists of space separated codes in each line.

So, there is a need for domain specific language, flexible enough to describe the most of time table requirements, while easy enough to be understood.

IV. Timetabling description language

The design of such language is driven by desire that description of timetables consists of sentences composed in the language similar to natural language. Sentences can be prepared to be grammatically correct in that natural language, but some words can be omitted in order to reduce typing. This EBNF grammar of the language used to describe the timetable is adapted for English.

```
SCHEDULE   = {sentence} EOF .
sentence = (identifier ( ["is"] ["a"]  (dDay | dTime |
dRoom | dClass |  dSubGroup | dAgregate | dTeacher |
dCourse |  dUnavailable | dLocate) | dTeacehs | dConflicts |
dWishes | dAvoids )) | dFavorize  "." .
dDay="day" ["of" "the" "week"]["number" number].
dTime=["time"] "slot" ["number"] number "on" identifier
["starting" ["at"] string] .
dRoom=("room" | "set" ["of"] number ["rooms"]) [[","]
"called" string] .
dClass="class" ["called" string] .
dSubGroup="subgroup" ["of"]  ["class"] identifier [[","]
"called" string]  .
dAgregate="composed" ["of"] identifier  {"," |  identifier |
"and"}  [ "called" string]  .
dTeacher="teacher" ["called" string] .
dCourse="course" ["called" string] .
dUnavailable="unavailable" ["on"] ["time"]
[("slot"|"slots")] identifier {(","|"and")   identifier} .
dLocate="located"["at"] identifier{(","|"and") identifier} .
dTeaches="teaches" {  ["course"] identifier  |
 "to" ["the"] ["class" | "group" |  "composition"] identifier
number ["times"] ["a" "week"] |  ("preferably"|"exactly"
|"not"|"avoidably"| ) "on" ["the"] ["time"] [("slot"|"slots")]
identifier["(" number ")"] |   "in" ["the"]("class" "room"|
"teacher" "room"|"C"|"T"| ["room"] identifier) |  "with"
number ("double" | "doubles" |   "triple" | "triples" |
"quadruple" | "quadruples" )["lessons"] |  "separately" |
"together" |  "," }.
dConflicts="conflicts" ["with"]  ["class" |   "group" |
"composition" |  "teacher"]  identifier .
dWishes="wishes" ["time"] [("slot"|"slots")] identifier "("
number ")"] {(","|"and")  identifier ["(" number ")"]} .
dAvoids="avoids" ["time"] [("slot"|"slots")]   identifier
["(" number ")"] {(","|"and")  identifier ["(" number ")"]}.
dFavorize="Favorize" { (  ("class" | "teacher" | "room")
("wishes" | "pauses" ["avoiding"]) |  "distance" "reducing"
|  "course" "balancing" |  )["(" number ")"] [(","|"and")]}.
```

This grammar allows sentences that describe the desired timetable to be generated or verified in very understandable way.

Cauchy is a teacher. Monday is a day of the week number 1. Einstein is a teacher called "Albert Einstein". M3 is time slot 3 on Monday starting at 8:00. Darwin is unavailable on time slots M3,M4 and M5. Iae is a subgroup of Ia called "Ethic attendants". Comm1 is composed of Ia, Ib and Ic and called "Joint lecture". Einstein teaches course Math. Sartre teaches French to IIIa, 4 times a week, preferably on M4 (200), on M2, with 2 doubles

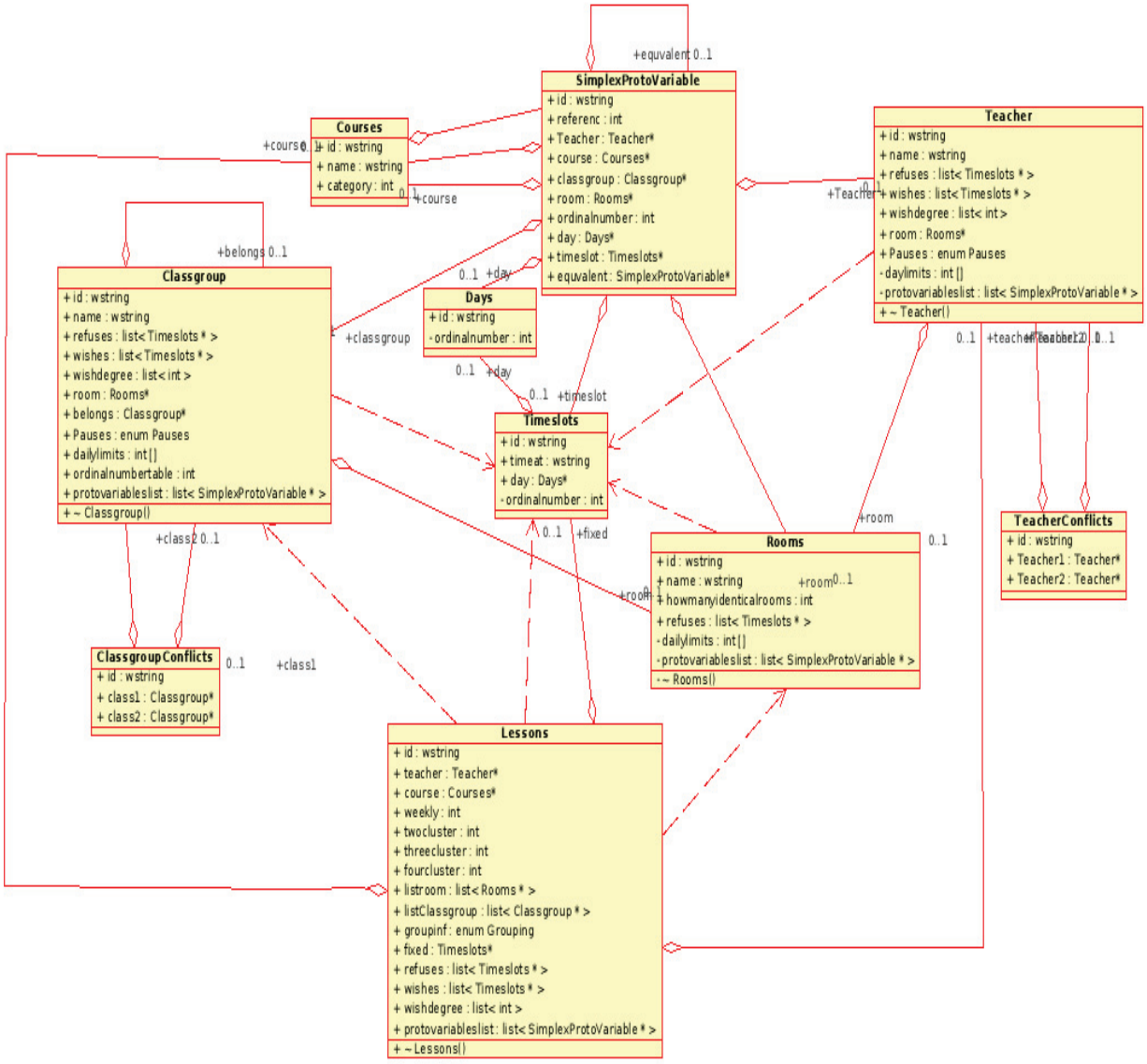

Figure 2: Class design

In the last two sentences we have constants given in parentheses which optionally give intensity of preferences. Some sentences can be written also in shorter form:

A118 room. Sartre teaches to IIIa, 4.

In addition to the sentences which describe the desired schedule, the DSL may include some sentences which give hints about choosing the optimal schedule among different ones, or the sentences which are specific to the used algorithms. They are not given in a grammar above, as they are algorithm specific. Example of such sentence is

Try 20 times randomly.

V. Implementation Details

To implement such DSL we used the parser generator Coco / R [7], because it generates easy to read parser, C++ compiler and library for solving problems of linear and integer programming GLPK [8].

The parser generator solves a problem of parsing, the GLPK library solved all the equations, but the generation of the equations need to be programmed manually. The first step after syntactic and semantic analysis is to create collections of teachers, classes, groups, rooms, days, courses, class conflicts, lessons etc.

The collections are here given as C++ classes representing the elements of a collection, shown in UML diagram on Figure 2.

A Classes Days and Timeslots represent the day of week and time slot within a day. Time slot points to the corresponding day.

The class Course describes a course. Property “id” contains short name of the course, while property “name” contains long name. Property “category” is used to classify the course, to avoid two similar courses in a day.

The class Rooms keeps an information about available rooms. It can connect a list of time slots when the room is not available (here called refuses). Each forbidden time slot reduces daily limit by 1. If the school is organized in a way that it does not matter in which room the course occurs, and it is just important that we have enough available rooms, then we can use the property which mentions count of identical rooms.

The class Teacher keeps data about every teacher in the school, her preferred and undesired time slots, forbidden time slot, list of preferred rooms, pause policy etc. The

similar properties are also present in the class ClassGroup describing group of students, which might be member of some larger group. Classes TeacherConflicts and ClassGroupConflict describe pairs of teachers or student groups that can not have lessons at the same time.

The class Lessons describes every teaching unit (teacher, course, student group, already predetermined time slot, number of lessons per week, lesson distribution in week, preferred and undesired time slots, forbidden time slot, list of preferred rooms etc).

The class SimplexProtoVariable is basically class Lesson dissolved to single occurrence. For example, if the lesson is taught 4 times a week, we will have one instance of the class Lessons and four instances of the class SimplexProtoVariable.

To make it practical to generate the equations that generate the timetable, some of the classes described above are not mere descriptions of entities, but they also point to the related elements. For example, the list of rooms contains description of each room, but every element of this room might contain list of pointers to the time slots when the room is not available, as shown on the Figure 3:

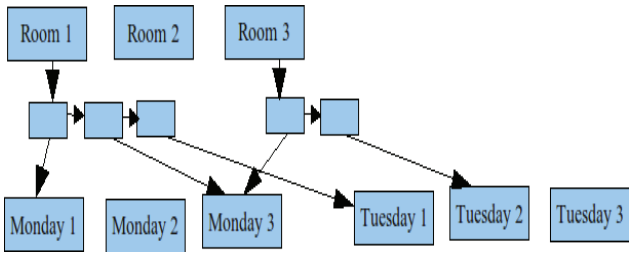

Figure 3: Structures linking

The most of other classes have the collection pointing to the elements of the list of SimplexProtoVariable where this class is referenced as well. Such approach significantly speeds up equation generating.

The next task is to use the parsed data in several ways. It can be converted to programs in language for combinatorial optimizations, into set of equations, or used for checkers of manual timetables etc.

To make sets of equations, the program creates a set of binary variables from the meaningful combinations of teachers, classes, objects, rooms, courses and time slots. They essentially determine whether the lesson is held in the specific time slot or not. For example, the variable EinsteinPhysicsIIaC1R12Monday should have a value of 1 if Einstein taught a first weekly lesson in physics to the class IIa in room R12 on Monday, and 0 if not. This set of variables is essentially Descartes product of SimplexProtoVariables collection and Days collection.

After the generation of variables we have equation and inequality constraints that define the problem constraints. The set of equations is huge, but the generation of each equation is only little more than visiting attached list of SimplexProtoVariables. For example to ensure that Bohr will teach Physics in Ia, room R15, for a second time of week in one of working day, sum of all binary variables must be equal to 1, as shown in the following equation.

BohrPhyIaR5C2Fri + BohrPhyIaR5C2Thu + BohrPhyIaR5C2Wed +BohrPhyIaR5C2Tue + BohrPhyIaR5C2Mon = 1

Our approach which uses two-phase linear integer programming to solve the problem reduces the required computation time, by decomposing the problem to determine the day and then, in the second phase, to generate a daily schedule. This job is repeated by perpetual improvement of the goal function using evolution strategy. It is in more detail explained in [9]. The solving of the first set of equations and inequalities will determine that the course will be taught at specific day and then for each day in a week a next sets of equations will be repeated.

VI. Future Directions and Conclusion

Due to similarity to natural language, this DSL should not be limited to English, actually we have already developed sentences in the local language of the users. However, there is much more room to improve the DSL. Due to different scholar systems, there might be many specific situations in timetable requirements, which needs to be somehow explained. For example, some schools operate on fractionated time slots, and some try to offer big number of elective courses to everyone. Of course, speeding up the existing ones, combining of them, and inventing the new ones can always improve the timetable algorithms, although this is not related to DSL.

This paper covered one proposal for domain specific language, which describes the desired timetable for school scheduling. The language is designed to be easily understood, but still parse-able. With good data structures, it is possible to generate a set of equations that generate the timetable automatically, or can be used to check the manually generated timetable against the constraints.

References

[1] Frisch M. A, Harvey W., Jefferson C., Hernández M. B. and Miguel I.. (2008) ESSENCE: A constraint language for specifying combinatorial problems. Constraints, 13(3), Springer

[2] Fukunaga A, Rabideau G., Chien S, and Govindjee A. (1997) ASPEN:An application framework for automated planning and scheduling of spacecraft control and operations. International Symposium on Artificial Intelligence, Robotics and Automation in Space Tokyo, Japan, (pp 181–187) , iSAIRAS

[3] Marriott K., Nethercote N., Rafeh R., Stuckey P. J. , de la Banda M. G. , and Wallace M. (2008). The design of the Zinc modelling language. Constraints, 13(3): (pp. 229–267.), Springer

[4] Van Hentenryck P and Michel L, (2003) The Modeling Language OPL — A Short Overview, Optimization Software Class Libraries, Operations Research/Computer Science Interfaces Series, 2003, Volume 18, (pp 263-294), SpringerI

[5] Schafhauser W. (2010), TEMPLE - A Domain Specific Language for Modeling and Solving Real-Life Staff Scheduling Problems, doctoral disertation, Technischen Universität Wien, Fakultät für Informatik

[6] International Timetabling Competition ITC2007, (2007) data file format sample, Available at http://www.cs.qub.ac.uk/itc2007/curriculmcourse/initialdatasets/comp03.ctt

[7] Mössenböck H., (1990) A Generator for Production Quality Compilers, ETH Zurich,

[8] Makhorin A. (2010), GNU Linear Programming Kit reference manual, Department for Applied Informatics, Moscow Aviation Institute, Moscow, Russia

[9] Ribić S. and Konjicija S., (2011) Evolution strategy to make an objective function in two-phase ILP timetabling, Telecommunications Forum (TELFOR), 2011 19th, Belgrade