A domain-specific language in course scheduling

This project is inspired by my experience in high school. The administrative staff scheduled 11 courses for about 30 classes. Then every teacher checked the timetable to make sure there was no collision and request some adjustment according to his/her preference. It was a heavy workload to collate the whole timetable for large numbers of classes.

A domain-specific language focusing on the course scheduling can make it easy to evaluate different constraints and produce the result. In this project, I try to develop a DSL specified to schedule courses and automatically validate the timetable.

## 1.Metamodel

A timetable is a result of course scheduling. In a timetable for the whole school, there are sub-timetables for different classes. I use two classes Timetable and SubTimetable. The use of one class Timetable and a reference with containment to itself can also achieve this goal. But the problem is that the user can make a new child Timetable under any timetable. That may make people confused.

In the timetable, we need to define the school time which the course should be limited in. A sub-timetable consists of seven columns that represent every day in a week. Here, I define an Enum datatype of “day” (as shown in Figure1). Then, in the day column, courses are listed as the time sequence. We should show the name, time, instructor and room of a course. The time of a course can be set with the start time and duration. The end time of the course will be automatically calculated. As for the attribute “classlimit”, it is the limited number of the course that should under the capacity of the room chosen. The sub-timetable is for one class, so the student number in a class is fixed. Designating this attribute “classlimit” to Class SubTimtable is also an alternative.

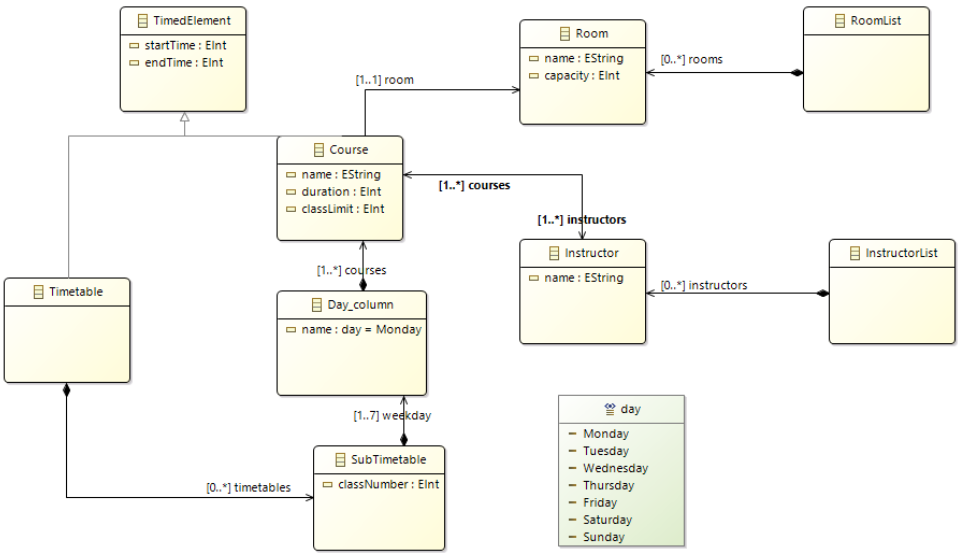


*Figure 1 Class Diagram of the Metamodel*

Above are the basic elements showed in a timetable. The rest are two elements related to the course. Besides the “Room” and “Instructor” two classes, there are two classes “RoomList” and “InstructorList” used to contain them respectively. Why are these two classes defined? That resulted from the feature of a tree-based editor which I determined to build. For example, we have to define an instance of course and an instance of room to fulfill the reference between these two classes. It would be better to use a RoomList to contain if there are lots of Rooms.

For the class Instructor, I declare one attribute of name and a bidirectional reference of courses. Because an instructor may teach more than one course and one course may be taught by several instructors, I make the multitude of the reference between course and instructor more than one.

Also, I have considered another choice of the metamodel. We can use a class CourseList like the RoomList and InstructorList containing the class Course. We use a class CourseSlot to replace the position of the Course in use now. The CourseSlot has a bidirectional reference to the Course. It seems to be better than the current metamodel. But I don’t try this idea because I have finished most work.

## 2. Generated Tree-based Editor

In this project, I use an Emfatic file to generate a dedicated tree-based editor conforming to the metamodel. Compared with the text-based editor and GMF editor, the tree-based editor is not convenient to edit and check the content. But it can explicitly and structurally reflect a timetable’s hierarchy.

Firstly, we should make the instances of class Timetable, InstructorList, and RoomList.

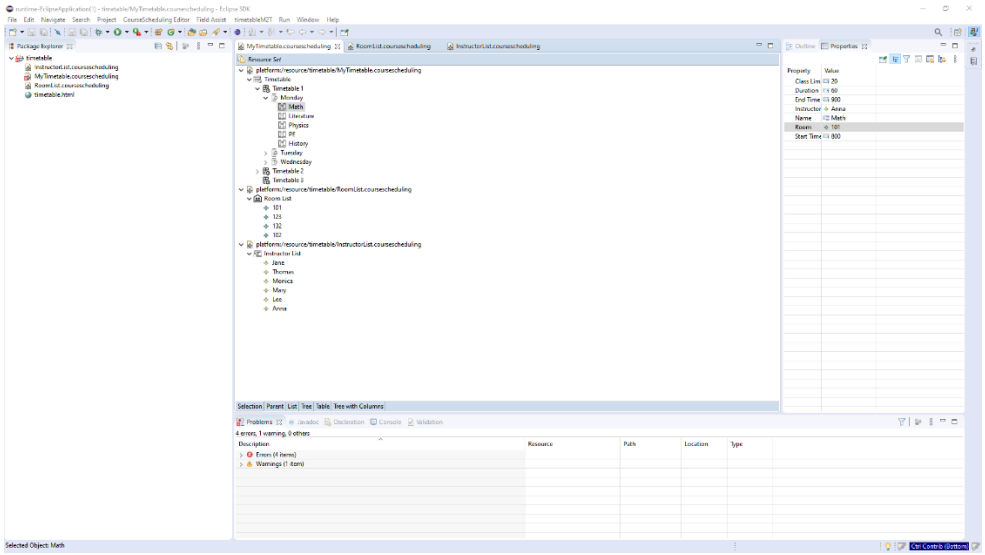

*Figure 2 Generated Tree-based Editor*

In the RoomList, we create a new Room and assign value to the attributes of “Name” and “Capacity”. In the InstructorList, we create new Instructor with an attribute of name and a

reference of courses. The reference of courses need be assigned after we instantiate courses in the Timetable.

After finishing these, we need to make the Timetable refer to them by loading resources. Thus, we can choose the instructors and room for a course.

Now, we can work on the timetable. There are only two attributes “Start Time” and “End Time” for the root “Timetable”. The time is in the format of 24-hour. If the start time of school is 8:00 and end time is 16:30, we should input 800 and 1630 respectively.

Under the Timetable, we create a new child SubTimetable with Class Number. Next, we create the Day_column of the sub-timetable, i.e. the days when the courses are arranged in a week. For this node, we need to select the weekday. Then, we can create a new child Course to begin doing course scheduling in one day below or continue to create sibling Day_column.

For the node Course, the only task is to fill the blank properties. As discussed in the first part Metamodel, we don’t need to manually fill in the endTime, which are auto filled in based on the startTime and duration.

The tree-based editor is easy to get started. Anyone knowing about a tree structure and some operation of eclipse can work well on this editor. But it is indeed tedious to operate. Users have to click, choose, and input continually. It can reflect the structure of a timetable, but it can’t show all the information in one window. We can’t quickly get the information about other courses to determine the suitable arrangement for the course we are editing. This may be the biggest problem.

3. Epsilon Validation Language

To validate the timetable, I made several constraints as follows.

(1) Blank property

This is to help find the element with blank properties. In the EVL file, there are several constraints for this situation.

The property classNumber of SubTimetable is undefined. This result in that we can’t confirm which class this timetable is for.

There are 6 properties in the Course. It is easy to forget to input the content required. To help locate the blank, I define two context-defining operations to locate the sub-timetable and day column which the course belongs to. The bodies of these two operations are the same except for the return values.

(2) Time format

As mentioned above, the time format is 24-hour. But I didn’t make any constraint in the editor. For example, separate hour and minute blank or use a dropdown to choose.

Therefore, I use a constraint here to test whether the input time is qualified. Because the input time is an integer, I check the time format by testing if the first two digits are between 0 and 24 and the last two digits are between 0 and 60.

Here I use a context-less operation to apply to check the format of the time inputted in the Timetable, SubTimetable and Course.

(3) Repetitive day

The day column in the sub-timetable has the default value "Monday". So, the sub-timetable would have day columns with the same name if the user forgot to modify. We can check if the number of the property "weekday" with the same value is not more than 1.

(4) Wrong time

In the context timetable, the startTime must be smaller than the endTime.

Meanwhile, due to the endTime of a course is automatically calculated bases on startTime and duration, the endTime is adequate. However, the start and end time of a course should be in the range of the school time, i.e. between the startTime and endTime of Timetable. I checked this time constraint in the context of Course.

(5) Overlapped course time

In the same day, the latter course must begin after the previous course is over. The constraint can be validated if there was not a course which startTime was less than the endTime of the previous course. I choose to implement this constraint in the context of Day_clomn. It seems to be easier in the context of Course.

(6) Room issue

In the timetable, the capacity of the room scheduled to the course should not overpass the number limit of the course. Also, I tried a solution to fix this issue by changing to another room with adequate capacity and not occupied at that time.

Another room issue is to check whether the room is occupied at the same time. This issue only happens across sub-timetable. Firstly, I try to find all the courses with the same time slot in one day. Next, I check whether there exists the same room in these courses. Eventually, we can get the result which course's room is occupied.

(7) Instructor with two courses

One teacher can't teach two classes at the same time. As the implementation as the occupied room in the previous constraint, we can validate the instructor scheduled for a course.

In this project, I followed an article about live validation in GMF-based editors with EVL[1]

[1] Live validation and quick-fixes in GMF-based editors with EVL. (n.d.). Retrieved March 19, 2020,

to integrate the custom EVL into my editor. I create a plugin project and add *org.eclipse.ui.ide* and *org.eclipse.epsilon.evl.emf.validation* to the list of dependencies. Next, in the Extensions tab, I add the *org.eclipse.epsilon.evl.emf.validation* extension, and in the namespaceURI field of the extension set the value to courseScheduling and in the constraints field select the scheduleChecking.evl EVL file. Then, I add the *org.eclipse.ui.ide.markerResolution* extension and create a *markerResolutionGenerator* with the following details:

- *class : org.eclipse.epsilon.evl.emf.validation.EvlMarkerResolutionGenerator*
- *markerType : org.eclipse.emf.ecore.diagnostic*

Now, we can use the validate button built-in Eclipse to run the EVL plug-in.

4. Model-to-text transformation

After we finish the course scheduling, we can use the menu button "timetableM2T -> generate html" to produce an html file to demonstrate the timetables for different classes.

Timetable: Class 1

| Time | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday | Sunday |
|---|---|---|---|---|---|---|---|
| 08:00-08:30 | Math 08:00-09:00 [Anna] Room 101 | Literature 08:00-10:00 [Mary] Room 102 | Math 08:00-09:00 [] Room 101 | | | | |
| 08:30-09:00 | | | | | | | |
| 09:00-09:30 | Literature 09:00-10:00 [Mary] Room 101 | | | | | | |
| 09:30-10:00 | | | | | | | |
| 10:00-10:30 | Physics 10:00-11:00 [Thomas] Room 132 | | | | | | |
| 10:30-11:00 | | | | | | | |
| 11:00-11:30 | | | | | | | |
| 11:30-12:00 | | | | | | | |
| 12:00-12:30 | PE 12:00-13:00 [Lee] Room 123 | | | | | | |
| 12:30-13:00 | | | | | | | |
| 13:00-13:30 | History 13:00-15:00 [Monica] Room 102 | | | | | | |
| 13:30-14:00 | | | | | | | |
| 14:00-14:30 | | Music 14:00-14:30 [] Room --- | | | | | |
| 14:30-15:00 | | | | | | | |
| 15:00-15:30 | | | | | | | |
| 15:30-16:00 | | | | | | | |

Timetable: Class 2

| Time | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday | Sunday |
|---|---|---|---|---|---|---|---|

*Figure 3 Demo of timetables*

I create a plug-in project to run a code generation in the editor by reference to two examples in the epsilon website and one built-in template in eclipse. "Hello, world

from https://www.eclipse.org/epsilon/doc/articles/evl-gmf-integration/

Command" template shows how to add a customized menu to the menu bar. "examples.egl.library"[2] example is about generating code from XML and running the code generator from Java. The above two examples can help achieve the goal of running code generation in the instance of eclipse. Now the rest is to load the EMF model and ecore from Java. The example "example.standalone"[3] show how to running a standalone Java application to execute different languages.

The editor can invoke the timetableM2T.egx EGX file to scan the objects in the model to generate the cell content in the timetable. Because the table in html is print by line after line, we can't generate the code as the nature day column in the model. The solution I figure out is to scan the courses in different day columns from Monday to Sunday by iteration.

Here, I assume that the start time of the timetables is the hour or half hour and the unit of a timeslot is 30 minutes. Due to the time stored in the model is an integer, I write an operation to format the integer in EOL to make it easy to read. In the process of code generation, we need to detect whether there is a course in the sub-timetable fit in the current timeslot. If so, we need to calculate the row number of the course and generate the corresponding html tag. Also, we need to notice that a blank cell should be reserved when the next row is being generated if it is already occupied by a course. Every sub-timetable is produced by the row scanning of the model.

Currently, the html file is in the plugin directory. After it was moved to the workspace, the code generation is done.

Now, users have to add instructors and rooms in the tree-based editor. This operation is not as convenient as in a text-based editor. If we could add a model-to-model transformation module in the DSL to get models from text files like XML generated from EXCEL, it would be better.

[2] https://git.eclipse.org/c/epsilon/org.eclipse.epsilon.git/tree/examples/org.eclipse.epsilon.examples.egl.library

[3] https://git.eclipse.org/c/epsilon/org.eclipse.epsilon.git/tree/examples/org.eclipse.epsilon.examples.standalone/src/org/eclipse/epsilon/examples/standalone