A Open source timetable production system for courses and examse

Ruben Gonzalez-Rubio · Balkrishna Sharma Gukhool

Received: date / Accepted: date

Abstract Diamant is a software system used to manually or automatically produce course and exam timetables at the Université de Sherbrooke, where it has been in use since 2001. We have decided to publish Diamant in Summer 2012 as an open source distribution. To be useful to other institutions, the source code must be modified as little as possible to facilitate the integration of new algorithms and new input/output formats. We present here how the program is organized and where modifications will take place.

Keywords University Timetabling · Design Patterns · Diamant

1 Extended Abstract

Producing course and exam timetables with Diamant at the Université de Sherbrooke has been done now, for more than 10 years, in different faculties. The course timetable can use post-enrolment mode or curriculum-based mode. We have decided to publish Diamant as an open source project to provide access to other universities that could be interested in using it. We think that, in some cases Diamant can be used as it is. The report [Gon07] is a detailed account of how timetables are produced at the Université de Sherbrooke, and some of its external characteristics were also presented in [Gon10].The timetables can be produced manually or automatically.

Diamant was designed by our research group $\epsilon\chi$it and the work was done by various developers. It was developed in an iterative way, and functionalities were added according to the users demands. The system is divided into two parts: one that takes the

Ruben Gonzalez-Rubio
Directeur du $\epsilon\chi$it Lab
Département de génie électrique et de génie informatique
Université de Sherbrooke
Tel.: +819 821 80 00 x 6 29 31
Fax: + 819 821 79 37
E-mail: Ruben.Gonzalez-Rubio@Usherbrooke.ca

Balkrishna Sharma Gukhool
Dpartement de génie électrique et de génie informatique
Universit de Sherbrooke

data from a database at the University, and formats the data in a way that Diamant can read. The second part is using Diamant to produce the timetables. Here, we present only Diamant, which is written in Java. In some cases, the software could be adapted to particular needs by taking the source code and modifying it. However, the problem with that approach is that there will be n different versions of the code. A second approach would be to apply, as much as possible, the open-closed principle, which means that the code can be open to modifications by adding new classes but closed to other modifications. This could be achieved by using the design patterns. The advantage is that the code can be customized without having different versions, the objective being to add only new classes. The classes needed for an implementation will be instantiated at run time.

At the core of Diamant is the MVC pattern (Model, View, Controller); the Model is a class containing a set of events, a set of instructors, a set of rooms and a set of students. Furthermore, the Model contains also a timetable which consists of slots (Periods), where Events will be assigned. The Model keeps the state of the timetable. The Views give the user a visual representation of parts of the sets or the timetable. The Controllers are there to take user interactions to change the Model, and when the Model is updated, the View is also updated and the user can see the change on the screen. The raw data used in the sets are only integers and strings, which helps to display them in the Views. Manual operation is carried out by opening a dialog, from a menu, the dialog is in fact a View, changing some data in the Model, and sending a feedback to the users in a View. Automatic operation is triggered by selecting an optimisation algorithm from a menu. The algorithm makes data assignments avoiding conflicts and optimising when possible, and when no more optimisation is possible, the algorithm quits and the Model is updated. The Model data is read from a file and written to a file, that preserves the state of the Model, which can be the final state when the timetable is finished.

Before publishing Diamant as it is, we are conscious of some of its disadvantages :

- First, that the code was implemented to satisfy the Université de Sherbrooke needs.
- The code is not homogenous, as it was written by various programmers.
- Finally, the system is in French only.

The last disadvantage is easy to change. To offer the application in other languages, we think that we must provide resources to facilitate internationalization; we need to have external messages in English and in French, as well as examples of how to work in another language. The messages of each language will be stored in a language resource bundle. So, normally no modification to the code is needed.

The first two disadvantages may need a large refactoring of the program. This refactoring will be done in an iterative way.

Our two goals before doing the refactoring are:

- We must offer an program that can be used by other developers, in principle they will add only new classes.
- The refactoring must reduce the class coupling, which is high in the current version.

These two goals are compatible. Diamant will be published as it is and new versions containing the refactoring work will be published.

In the next section, we present some details of the system to illustrate how it is organized.

Details of the work

Basically, a timetable is prepared from data concerning the timetable: the timetable itself, the events, the students, the instructors and the rooms. At the end, each event is assigned to a Period in the timetable, for each event a room, an instructor and a set of students are assigned. When the work is done manually, the assignments are done by indicating each assignment via the GUI. For example, in a dialog box, it can be indicated that an event will take place Monday at 13h00, and another Wednesday at 14h00, and so on. In all dialog boxes, there are list of values where the user can select one. By design there are no text boxes where any value can be typed. When the work is done by the optimisation algorithm, it will allocate automatically all possible assignments.

When trying to produce a timetable, conflicts can arise; so the system must indicate this. In the case of automatically producing the timetable, the program tries to reduce to a minimum the number of conflicts.

To explain the program, we distinguish the following parts:

- The input and output of data.
- The GUI and the manual data manipulation
- The optimisation algorithms

1.1 Using the design patterns

A design pattern is a general reusable solution to a commonly occurring problem within a given context in software [GHJV95]. We are going to refer to some of these design patterns.

We already mentioned that Diamant is organized using the MVC pattern (Model, View, Controller). The Model is a big class it contains the data representing the state of the timetable, those data are divided in classes SetOfX[1] and a Timetable. The sets of instructors, of rooms and of students are used as information that help decisions when a timetable is produced. For example the characteristics of a room (size among others) can help to assign the room to an event. The sets of events and the time table are used to keep the state of the Model when a decision is taken.

There are various Views and various Controllers. The Views give the user a visual representation of parts of the sets or the timetable. The Controllers are there to take user interactions (a external change in the View) and send them to the Model, and when the Model is updated the Views are also updated, and the user can see the change on the screen.

For example in a View, that can be a dialog, it is possible to assign a instructor to an event. That can be done by selecting the instructor name for this event in the View, when the button "ok" is pressed the action takes place, that means that the Controller sends the name of the instructor to be assigned to the corresponding event in the Model where the change takes place.

The Model gets its data from a file and at the beginning sends its current state of the model to a file when some work is done. This work can represent the final state or an intermediate state.

[1] Where the SetOfX could be a set of events, a set of instructors, a set of rooms and a set of students.

The DTTStructure represents the timetable with its Model organized hierarchically Figure 1. It can contains c cycles, each cycle contains d days (normally 5 days), that can be seen as a week, in a day there can be s sequences, and each sequence can contain p periods. There is no overlap in periods contained in a sequence, and there is no overlap in sequences for a day. With this organization, it is possible to offer a very flexible way to define a timetable. The periods can be seen as slots where events are assigned.

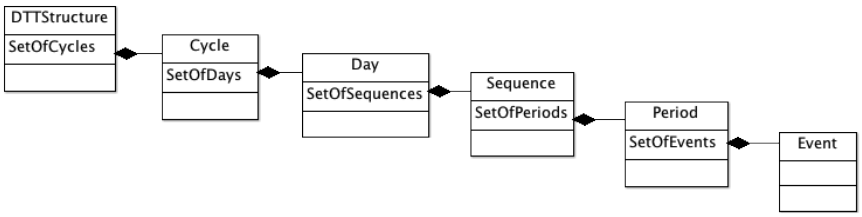

Fig. 1 The timetable hierarchy

Where the sets are lists of Objects that are defined for the type of its constituents.

1.2 The input and output of data.

The raw data (input and output) used in the sets are only integers and strings. This helps to display them in the Views and a developer can read the files if needed.

We can have the following scenarios with the data:

- The data used in an other university are the same as that of Diamant, then there is no work to do.
- The new data can be mapped onto the Diamant data; maybe some attributes have another name. That can be done by reading the new data and put the data in a file that can be read by Diamant.
- The new data are a subset of Diamant's data, that means that some of the Diamant data must be initialized to safe values; these values are safe when there are no exceptions and the optimisation algorithm does not use these values.
- The new data contains new attributes, in which case these new attributes must be part of the data.

The first three scenarios are easy to deal with. The last one, however will need the usage of the Strategy pattern, Figure 2. In that pattern a Diamant file (one for each type of data) can be read by one class, namely DiamantFormat and a different format can be read by another class NewDataFormat. The readFile is a variable of type ReadFileBehavior and each class in the Model has one. That means that the readFile method can be fixed at runtime. The interface ReadFileBehaviour contains one method readFile. The classes DiamantFormat and NewDataFormat implement this method, so the variable readFile is used to call readFile without knowing which class is instantiated. For each new format only new classes will be added.

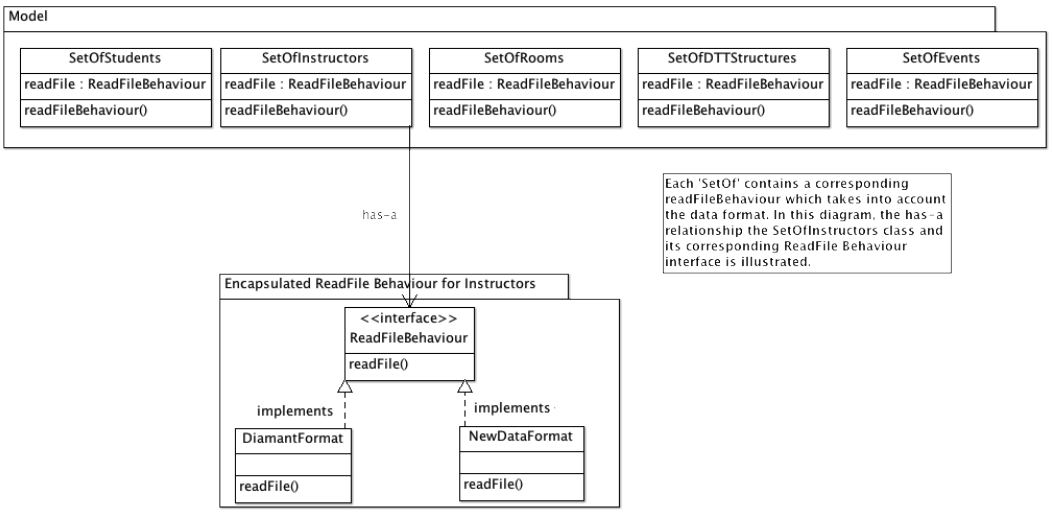

Fig. 2 Strategy pattern for reading files

1.3 The GUI and the manual data manipulation

The Model contains two types of data: immutable and mutable. The mutable data are used to make assignments (see next subsection). All the other data in the Model are immutable, that means that they do not change during program runtime. For example, it is not necessary to change an instructor name by Diamant to produce a timetable. If any change is needed on immutable data, it must be done outside Diamant. For example, if an instructor name has an error, it must be changed somewhere else, possibly in the database, then the files can be reloaded into Diamant. To add attributes to the Model, there are new PropertyLists containing couples with each couple having the name of the attribute and the value. So for example, in a University they add an attribute sex for students. This can be done by adding a list of attributes to the students in that list the new attribute will be a couple with the String sex and the value M or F. This information can be used to set groups with an appropriated mix of students. The new attributes must be also immutable. All the dialog boxes will implement interfaces and this allows the addition of a new dialog box which can display the new attribute. Again, modifications to offer new functionalities will be done by adding new classes.

1.4 The optimisation algorithms

The timetable is built by assigning events to Periods and assigning data to Events. In fact, to make an assignment, Periods and Events contain instance variables, which are lists. For example to assign an Event to a Period, the Event is added to the list; when the Event is removed from the list, then the assignment is no longer valid. Manual operation allows the assignment process by using dialogs, where choices are made by selecting a value from a list and this changes the model. Automatic operation is started by a menu where an optimisation algorithm is defined; when started it works on the

Model. It makes assignments and removes assignments to minimize conflicts. When the algorithm finishes, the View is updated, showing the current status of the timetable.

Also different optimisation algorithms can be implemented by a strategy as illustrated by Figure 3.

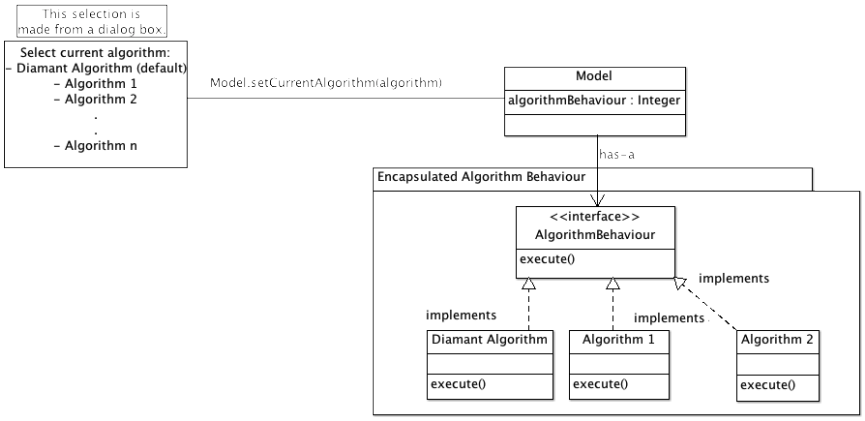

Fig. 3 The strategy pattern for optimisation algorithms

Each strategy can be instantiated when a menu item is activated; if the menus are implemented by the command pattern, then when a new command is needed a new class is written and the menu can be added to the configuration.

In some cases the algorithm could need the data in a different data structure, for example instead of using the timetable of Diamant a simple matrix can represent the days and the periods. That means that the data in the Model must be mapped onto new structures, so the algorithm will be divided in three steps : mapping the Diamant Model onto the new structure, the optimisation algorithm and mapping the result onto the Diamant Model.

Conclusion

We think that the system Diamant could be used and extended for other universities. Customisation can be done in a simple way without having to rewrite large parts of the system. We will be glad to help other institutions to adapt Diamant to their special needs. Finally the diamant URL is:
http://tictacserver.gel.usherbrooke.ca/exitopenprojets/

References

GHJV95. E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

Gon10. R. Gonzalez Rubio. Diamant: A timetable production system for courses and exams. In *PATAT'10*, Belfast, Northern Ireland, August 2010.

Gon07. R. Gonzalez-Rubio. La production et la consultation d'horaires dans une université. Technical report, Université de Sherbrooke, 2007.