A timetable production system architecture for courses and exams

Ruben Gonzalez Rubio
Domingo Palao Muñoz

Département de génie électrique et de génie informatique
Université de Sherbrooke,
Sherbrooke, Québec, J1K 2R1
Canada
Ruben.Gonzalez-Rubio@USherbrooke.ca
Domingo.Palao@USherbrooke.ca

Abstract. The production of a timetable for courses and exams in an academic institution is a periodic activity, the frequency of which depends on the institution: yearly, semi-annually, or quarterly. A timetable must satisfy all the hard constraints and as many soft constraints as possible. Timetable construction is one step in timetable production. Since it is a very complex activity, the use of software helps carrying it out. In this article, we propose to analyze timetable construction from a broader point of view. We will consider the full range of timetable production activities from data gathering and data input to timetable delivery, including construction. We analyzed the information flow required for timetable production. It led us to understand the characteristics of such systems. Timetable production can be automated by using databases, a Web site, and timetable construction software. We propose a generic architecture of a timetable production system for courses and exams. We've developed that architecture around two fundamental qualities of software development: openness and extensibility.

Keywords : Timetable Production, Timetable Construction, Open Architecture, Extensible Architecture, Databases, Web.

1 Introduction

Timetable production is a periodic activity carried out in many academic institutions. Each year, they produce one or more timetables for their courses and exams. Computers can help significantly reduce production costs. In certain cases, automation starts with the use of timetabling software; in others, with the automatic processing of the data related with the work of the institution, for example the follow-up of students. Some software applications, such as DIAMANT

[Gon00] and SAPHIR [FF94], are designed to carry out timetable construction, but there are many others.

In general, timetable software uses one or more files as input, and one or more files as output. Working with files generates problems. It is necessary to build input files each time a timetable is produced. There are two ways of producing those files: manually or automatically. If the files are produced manually, typing errors can occur. Such errors can produce unexpected effects in the software. Furthermore, they are very hard to detect, because all the data have to be checked manually. In the other case, the files are produced in an automatic way. Then, the software must check the validity and the coherence of all the data.

Another source of problems, and even more expensive than the former one, is when a change is introduced into the system, either in the construction software, in the production software or in both. The change could originate from persons in charge in the institution, for example a new nomenclature for the buildings, a new teaching approach, or anything which can affect the way of creating groups, etc. The problem of modifying software is not solely limited to timetable production software; it is a general problem in software development.

The solution to the first problem rests upon the automation of data checking and data validation at the time of data acquisition, in order to ensure that the construction software will work with the right data.

The second problem is more difficult to solve. With a production software system designed to be modified (i.e. extensible), the cost of a modification is low, but if the system is monolithic and closed, any modification will be very expensive. A way of minimizing the effects of this problem is the development of software featuring extensibility.

In this article, we propose a timetable production system architecture for courses and exams having the following qualities: openness and extensibility. This architecture represents an easy solution to the two problems mentioned above. It reduces the cost of modifications, performs data checking and validation in precise stages of the process, and prevents redundancy of both code and data. This architecture makes use of a database, a Web site and a software for timetable construction. Data checking and validation will be made with the input and with integrity constraints in the database. The Web site is mainly used for data input, but at the same time, it can check the data validity, using the business rules before the data are recorded in the database. Another function of the Web site is to offer a personalized timetable, once it is built. The database allows data retrieval, according to various formats. Thus, the input of the construction software is simplified. Of course, with the assistance of the timetable construction software, it is possible to create a timetable with a minimum of conflicts.

Much research has been done in timetable construction: algorithms, mathematical formulations, and several softwares were proposed in PATAT conferences [BC97,BE01,BC03]. On the other hand, little effort has been devoted, until now,

to the overall development of timetable production. We think that the fact of viewing the process as a whole can contribute to the research in timetable construction. White [Whi00] proposed that the Web should be used for the diffusion of timetables. We are trying to go further by proposing to use the Web also as an interface for data input; moreover, we propose an architecture that can support support our ideas. De Causmaecker et al. [DPDV02] introduced the idea that the semantics Web can be used by researchers in timetable construction. They also introduced the idea of using XML as a communication language in timetable construction by agents. We also propose the use of XML, in order to facilitate exchanges between certain modules of our architecture. In Burke and al [BKP97] it is proposed to have a standard for the data format, which will be used by the timetable software; their goal was to compare algorithms (programs) in benchmarks. In a more recent work, Kingston [KL02] proposes the STTL language and an interpreter, with the same objective. Furthermore, Özcan [Özcan03] suggests another standard to define instances of timetable construction problems, using XML as a base language. It seems that it would be desirable to have a standard, but this goal is very difficult to reach. So our approach has been to use a database as a tool for extraction and data formatting. This way, the timetable construction software can receive the necessary data. Similarly, it would be possible to generate other instances of problems for other timetable construction softwares.

The development of our architecture was divided into two parts. We first analyzed the data flow in a generic timetable production system. Then, we defined the architecture, the basic blocks and the responsibility of each block participating in the architecture. Our objective of defining an open and extensible architecture was met during the project. This is presented in Sections 2 and 3.

Once the architecture was defined, we created an instance in order to implement that architecture in a timetable production system at the University of Sherbrooke. We reviewed existing technologies to make the right choices in accordance with the features of openness and extensibility. This is presented in Sections 4 and 5.

We will finish with an assessment and our conclusion in Section 6.

2 The Information Flow in Timetable Production

First, let us define the process of timetable production as a process which starts when the data for the timetable period P of validity are entered. This P period is a six-month, a quarter, or a year period[1]. The process ends when all the concerned people receive the schedule on paper or via the Web (see Figure 1)

[1] At the University of Sherbrooke we talk about a quarter, but actually it comprises sixteen weeks (almost four months), therefore the system must adapt to any P period.

and the period of validity of the schedule comes to an end. Multiple data are entered into the system, for example courses offered during period P, format of courses (3h, or 2h+1h), assigned or potential teachers for certain courses, assigned or potential classrooms, availability of teachers and classrooms, etc. We know that in some institutions, the students make a pre-selection while in others the timetable is made with no pre-selection. If any, the students' pre-choice is part of the data to be supplied. If there is no pre-choice, it is necessary to indicate the number of students enrolled in each course.

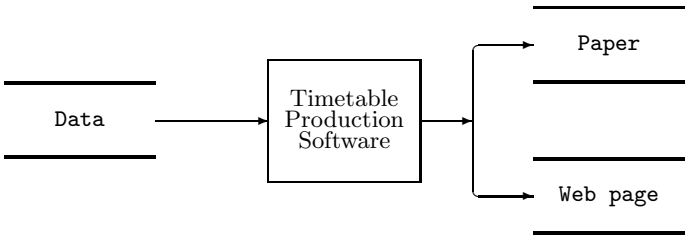

Fig. 1. Timetable Production

Figure 1 is an abstract view of timetable production, the system of timetable production including timetable construction software.

Timetable Production System. It is composed of several softwares, or modules, which work in an autonomous, but coordinated way. It is clear that the system must include two main components: the system to save data and the timetable construction software.

Data. Represents the data required to produce the timetable. We will save all the necessary data in a data back up system. In the figure, it is not indicated that these data can be handled by modules of the timetable production system. For example, the pre-choice acquisition module will save the courses followed by a student; the data construction module will take its inputs from the database, and, when construction is performed, the new data obtained will be saved in a data warehouse.

Paper or Web page. It is the output of the system once the timetable is built. This output can be saved in files to be printed on paper or as data to be presented on a Web page. In both cases, the data source is the database.

2.1 Data Preparation

Figure 2 shows data evolution during timetable production. The evolution presents instances of the data at different steps of the process.

The data warehouse can represent one or several databases. We can think of a distributed database. The choice of the database is an implementation decision. In a conceptual way it is only one data warehouse.

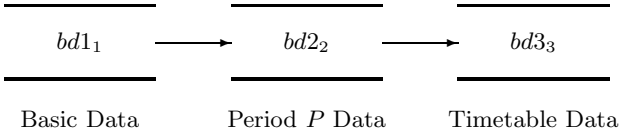

Fig. 2. Evolution of Data During Timetable Production

Basic Data. These data are defined as long-term data in the data warehouse. Information on the academic activities of a program, the programs offered, the teachers, the characteristics of classrooms, the students, all of them are examples of this type of information. In general, this information is shared by other applications. For example, a student is described by his (her) first name, last name, identification number, program, etc. An application using data on students is for example that which prints students grades on their evaluation sheet. Here, we are interested in the data associated with timetable production, which can be part of the complete institutional database.

Period P Data. These data are essential to the construction of a timetable for a given period. They contain the courses offered during that period, the availability of teachers for that period, etc. These data are a prerequisite for timetable construction, and will be entered before each construction. However, there will be data updates during the process. For example when an external teacher is assigned to a given course, we will enter his (her) name and availability in the database.

Timetable Data. These data result from timetable construction. They will contain all the information necessary to print or show a general and personalized timetable.

In addition to constructing timetables for courses, the system must be able to build timetables for exams. The evolution of data, in the case of exams, follows the same path as courses. With the exception that the data extracted from the database can be different: for example, for an exam timetable the teachers' availability may not be necessary.

There are two major activities in timetable production: data entering (including updates), and timetable construction.

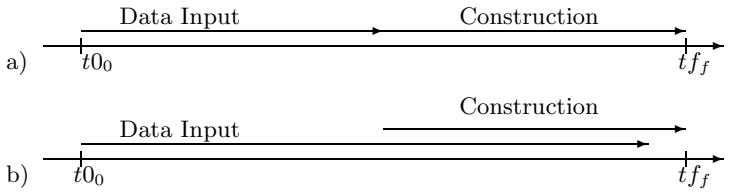

Fig. 3. Data Input and Timetable Construction Activities for a P Period

Figure 3 shows two cases of activities evolution. In case a), data input and construction are two activities carried out in sequence. This represents the ideal case. In case b) which shows the real world, data input (especially updates) is done at the same time as construction. At the beginning of construction, some data are fixed, but if the software or the person in charge does not find a satisfactory solution, it becomes necessary to make changes even in the fixed data.

The activities of data input and construction can be shared by many users. Each one has a role and associated privileges to modify a number of data.

Roughly speaking, we consider two types of users: the clerk and the person in charge of timetable construction. The clerk is the one who will enter the information into the database. The person in charge of timetable construction interacts with the timetable construction program.

The clerk will enter or update the data in the database. Valid data will be accepted and added to the database. For example, if the data expected is an integer number in a specific range and the input value corresponds is within the set values, it will be entered into the database. Otherwise, the system will force the clerk to enter a new value.

Timetable construction is an activity of an iterative nature. Generally speaking, the algorithms of timetable construction propose meeting the hard constraints and trying to comply with the soft constraints as far as possible. Therefore, the person in charge will try to find several solutions from which to choose. In certain cases, it is possible to leave the program with unchanged data and find a new solution; but in other cases, it is necessary to change the original data to obtain a new solution. If it is necessary to make n tests, the solution selected by the person in charge will be solution s where $1 \leq s \leq n$. One has to

devise a way of working with the database in order to save the changes carried out for each iteration, and to make a "commit" when the solution is chosen and the timetable becomes final for the period.

3 The Architecture

Further to the study of data flows in timetable production, we are able to propose an architecture for implementing the system.

3.1 An Architecture for Timetable Production

Abstractly speaking, software architecture describes the elements of a system. It also shows the interactions between these elements, the models governing its composition and the constraints of these models [SG96].

Generally, when facing a complex problem, the best approach is to break it down into parts that become easier to solve with simple solutions. Then, when we combine all these small solutions, we can find the solution to our complex problem [BMR+96].

Of all architectures now available for the development of software applications, the one most appropriate to timetable production systems is that in layers.

The architecture shown in Figure 4 proposes dividing the system into three layers, each one with a well-defined function:

The Interface. Presents the data to the user, to allow data input and ensure exchanges with other layers.

The Business Logic Layer. Ensures data exchanges with the interface layer. Checks and validates the data input and sends the data to be presented in an adequate format. It also ensures data exchanges with the data persistence layer. Business rules are used to ensure the coherence of the system.

The Data Persistence Layer. Manages the physical storage of data in files with a certain format, or in a traditional database system, or in other persistence models able to manage complex databases.

3.2 The Three-Layers Architecture and the Timetable Production System.

For each element of the architecture we've introduced in the earlier subsection, we propose a component that is part of the timetable production application.

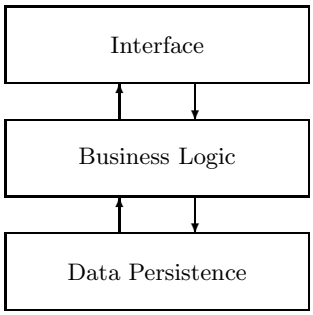

Fig. 4. The Three-Layers Architecture

In this subsection, we will show the integration between the data flow for a timetable production system and the suggested architecture.

The high-level design of our system of timetable production is simple (see Figure 1):

1. We defined three categories of data in a timetable production system. Figure 2 shows these categories. The interface layer treats this data as follows:
   (a) **Basic data.** This type of data is in a database belonging to the institution. The database may have its own interfaces for data input and display. The production system will use part of such data. An interface with other systems is required, and its responsibility is to seek the data required and save them in the production system. It's important to note that, for security reasons, the users of the system should not be allowed to modify the data of the institution.
   (b) **Direct Input Data for Period P.** This is performed by a clerk. Usually, this type of data changes with each timetable creation. To enter the data, the clerk has to complete fields on a Web page displayed by a navigator. At that point, the interface layer can make some minimal validations of data coherence, for example verify data type correctness, check that all mandatory fields are filled, etc. Depending on the size of the institution, there may be several clerks working at the same time in the application. Consequently, the necessary measures should be taken to allow concurrent work in the data persistence layer.
   (c) **Timetable Data.** Generated by the timetable creation software. This program is managed by the user in charge of timetable creation. Since this activity is of an iterative nature, the user responsible for creating the timetable must analyze the various versions of the work to find out which one is best suited. Once he (she) has found that version, he (she) proceeds with the "commit" operation to make the information available to the rest of users. Those are the output data from our system that are

sent to other systems or users. For example, a student will receive a personalized paper-based timetable or view the information on a Web site created to this end.

We should keep mind that the interface layer communicates only with the business logic layer. An application is unable to add, modify or erase data directly in the database. It must inevitably go through the business logic layer. This characteristic enables us to ensure the coherence of the data.

2. The data is treated according to certain rules and constraints. These rules are defined by each institution and they are coded in the business logic layer. It is here that we can express constraints; for example, the maximum number of students per group, the maximum number of courses a professor can teach, etc.
3. The data resulting from such processing must be stored in a place where it can be used after being generated. The data persistence layer is responsible for this activity. In a timetable production system, the data can be stored in a database (relational, object or other technology), an XML file, a text file or another storage media.

For composition elements and their relationship, see Figure 5.

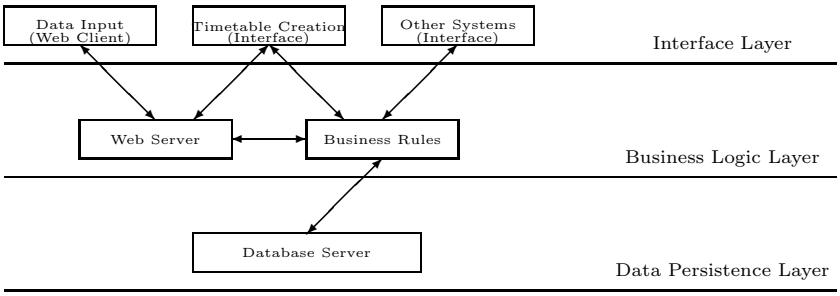

Fig. 5. Implementation of the Three-layers Architecture for the Timetable Production System

4 Technology Resources

To implement the architecture, we first need to find out what technological elements are required. The needs are as follows:

1. A language to facilitate data display by the user, and the creation of forms to enter data.
2. A set of rules to define how the data exchange is established with other systems.

3. A language to code the business logic.
4. A tool to manage data persistence.

Through our analysis, we found there is more than one option: there are actually several ways of implementing this architecture.

In Table 1, we show an open-ended list of possible instances.

Technology	Instance 1	Instance 2	Instance 3
Language for Display	HTML & JSP	Applet Java	Visual Basic
Format to Data Exchange	XML	Text	Text
Language to Code the Business Logic	Java	Java	Visual Basic
Data Persistence	Database	Database	Database

Table 1. Possible Instances of Architecture

For the construction of each instance, we put together the elements sharing common characteristics, or belonging to the same software family, and which are open.

For the database section, we did not mention any specific name of a database management system. The choice remains open to the implementation.

We have an open architecture, since it accepts several instances. The architecture is also extensible, because we could easily change the modules. For example, we can start with an element of data persistence like a file in text format, and then, we could make some changes to the interface element to be able to interact with files in XML format. Later, if we want to have the data persistence element evolve to a database system, it would be necessary to make changes in the interface layer and to program the data persistence element. We can make all these changes without touching the business logic layer. In Table 2, we show the possible evolution of an instance of our architecture.

Technology	Version 1	Version 2	Version 3
Language for Display	HTML & JSP	HTML and PHP	HTML and JSP
Format to Data Exchange	Text	XML	Database SQL
Language to Code the Business Logic	Java	Java	Java
Data Persistence	Text	XML	SQL, SQL Database

Table 2. The Evolution of an Instance of the Architecture

We can also imagine a system starting with an interface based in textual mode to show the information, but which will evolve with a display based in HTML and JSP by using a JSP server.

5 An Instance of the Architecture

Once we've established the architecture and we know that it is open and extensible, it is necessary that the instance we will implement be open and extensible too. We've shown three options of architecture instances (see Table 1). For our implementation, we chose option 1.

First, it should remembered that an element is open if it is built with approved standards, or if built with a private specification but made public by the developers[2].

It is important to remember that an element is extensible if it is easily adaptable to specification changes [Mey97].

Bearing these considerations in mind, we will now explain the analysis that brought us to our choice:

Language for Display. For the language for display and to enter data we chose HTML and JSP, because those languages are among the most widely used to create Web pages, and because we want to benefit from all the characteristics the Web site can offer for transactional applications.

Format to Data Exchange. To exchange data we chose XML. The XML language (eXtensible Markup Language) allows us to store, exchange and show data or information in a structured way. The language also makes it possible to handle, transform and visualize data with many formatting tools. For example, the information stored in an XML document can be displayed in a Web navigator. Moreover, when XML documents are stored in a database, they can be the object of requests and treated like any other data.

Language to Code the Business Logic. We chose Java. Java is an interpreted language that uses the Java Virtual Machine (JVM). There are several versions of this virtual machine for several different platforms, which means that the programmers can write for this language by using a platform, and then execute their programs in others.

Data Persistence. For the implementation of the data persistence layer, we chose to use a database. A database is an entity in which it is possible to store data in a structured way with minimum redundancy. This data must be used by programs written in different languages by resorting to ODBC standard technology (Open DataBase Connectivity). If we use Java as the language to establish a connection to the database, we can use JDBC (Java Database Connectivity). The standard JDBC is an application program (API) to establish the connection between the Java language and a large number of databases by using the "Write Once, Run Anywhere" philosophy.

Even if it is not the only option for the implementation of our timetable production system, we are convinced that our choice will enable us to have

[2] http://www.webopedia.com/TERM/O/open_architecture.html

the system evolve to other versions, adding or modifying new characteristics as required by institutions.

The system implementation seems complex. But, we can reuse several existing modules. This is one of the advantages of the opening and and the extensibility of the architecture and the instance. We can analyze the components by layers to know which ones can be reused, and which ones will be implemented:

Interface Layer. For this layer, there is a technological element which facilitates the work enormously, the Web browser. This tool is available in most computers with Internet access. It is necessary to centralize the efforts to develop Web pages with the quality of compatibility between two Web browsers: Internet Explorer[3], Netscape[4], etc.
The Internet site http://validator.w3.org/, offers a tool to test the compatibility of a site with the standard HTML. This tool is provided by the "World Wide Web Consortium"[5].

Businesses Logic Layer. There are two fundamental elements in this layer: a Web server and a server to deploy the business logic. Since we chose Java to code the business logic, we can use Apache HTTP[6] as a Web server and, and Jakarta Tomcat[7] as a server to provide the business logic.
The "Apache HTTP" project is an effort to develop and maintain an "Open source" HTTP server for more widely used operating systems like UNIX and Windows. The goal of this project is to provide a secure, effective and extensible server able to offer HTTP services according to current HTTP standards.
We will program the business rules in some small Java programs called servlets. A servlet is a Java program used to extend the functionalities of a Web server. A servlet is a program used to generate dynamic contents of the applications. It is an application carried out in the server which is downloaded dynamically when it is required. Jakarta-Tomcat is a servlet container.
We can say that the Apache and Jakarta-Tomcat couple enables us to make the deployment of a complete Web application. Besides, these tools are free.

Data Persistence Layer. For the data persistence layer, we said that we will benefit from the use of JDBC (Java Database Connectivity). The JDBC is based on the use of a pilot to give access to several databases, the advantage being that we can change the pilot according to the database supplier selected. As to the choice of the database, we said that it was open to the implementation, but the standards should always be considered. The latest standard published by the ANSI (American National Standards Institute)

[3] http://www.microsoft.com
[4] http://www.netscape.com
[5] http://www.w3.org/
[6] http://www.apache.org
[7] http://jakarta.apache.org

for the language which processes the data of a database is SQL-99[8]. This standard is used by the most important database developers like Oracle[9], Informix[10], IBM-DB2[11], PostgreSQL[12], and others. As to PostgreSQL, using the license is free of charge.

As we have seen in this section, we benefit from several open technological components which enable us to speed up development while keeping costs low.

6 Conclusion

We've presented the architecture of a timetable production system in an academic institution. An objective guided our approach: to have an open and extensible architecture so that it would ensure a long-life system.

We started by showing the needs for information exchange in the timetable systems in order to explain how the architecture is able to satisfy those needs. Moreover, in the presentation of the architecture, we showed that it has two qualities. Our architecture is open: it is possible to build several instances. The architecture is also extensible because we could easily change the modules. The responsibilities for each module are defined since it is possible to identify the part of the system that has been modified. It is necessary to continue developing, keeping in mind the objectives of openness (compatibility) and extensibility throughout the software development process.

It is now possible to answer the question: "Can the timetable construction take advantage of developments associated with timetable production?" Our experience in software development for the timetable construction software showed us that most problems come from the fact that the input data contains errors. The construction software must dedicate a large part of its code to detect those errors. Moreover, users spend time checking the coherence of the data before beginning the building process. The answer is then yes, there is a real benefit. By developing a production system, it is easier to ensure that the construction software has responsibility for timetable construction rather than that of checking the data. And it's easier to adapt the software to the changes.

Other benefits can also be considered, such as the user being able to consult the timetable on the Web, at home. That is especially well appreciated in countries where the temperature where sometimes drops to $-30°C$ and less.

[8] http://web.ansi.org
[9] http://www.oracle.com
[10] http://www-306.ibm.com/software/data/informix/
[11] http://www-306.ibm.com/software/data/db2/udb/
[12] http://www.postgresql.org/

References

[Mey97] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice-Hall, 2nd edition, 1997.

[Özcan03] E. Özcan. *Towards an XML based standard for Timetabling Problems: TTML*. 2003.

[Gon00] R. Gonzalez Rubio. Generating university timetables in a interactive system: DIAMANT. In *PATAT'00*, volume 1, Konztanz, Germany, August 2000.

[FF94] J. A. Ferland and C. Fleurent. SAPHIR: A decision support system for course scheduling. *Interfaces*, 24(2):105–115, Mar-Apr 1994.

[BC97] E. K. Burke and M. Carter, editors. *Practice and Theory of Automated Timetabling II, Second International Conference, PATAT 1997, Toronto, Canada, August 16-18, 1997, Selected Papers*, volume 1408 of *Lecture Notes in Computer Science*. Springer, 1997.

[BE01] Edmund K. Burke and Wilhelm Erben, editors. *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000, Konstanz, Germany, August 16-18, 2000, Selected Papers*, volume 2079 of *Lecture Notes in Computer Science*. Springer, 2001.

[BC03] Edmund K. Burke and P. De Causmaeker, editors. *Practice and Theory of Automated Timetabling IV, Four International Conference, PATAT 2002, Gent, Belgium, August, 2002, Selected Papers*, volume 2740 of *Lecture Notes in Computer Science*. Springer, 2003.

[Whi00] G. M. White. Constrained satisfaction not so constrainted satisfaction and the timetabling problem. In *PATAT'00*, volume 1, Konztanz, Germany, August 2000.

[DPDV02] P. De Causmaecker, Y. Lu P. Demeester, and G. Vander Berghe. Using web standards for timetabling. In *PATAT'02*, volume 1, Gent, Belgium, August 2002.

[BKP97] E. K. Burke, J. H. Kingston, and P. A. Pepper. A standard data format for timetabling instances. In *PATAT'97*, volume 1, Toronto, Canada, August 1997.

[KL02] J. H. Kingston and B. Yin-Sun Lynn. A software architecture for timetable construction. In *PATAT'02*, volume 1, Gent, Belgium, August 2002.

[SG96] M. Shaw and D. Garlan. *Software Architecture. Perspectives on an emerging discipline*. Prentice-Hall, 1996.

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture. A system of patterns*. Wiley, 1996.