GREN-Wizard: a Tool to Instantiate the GREN Framework

Rosana T. V. Braga[1]
Paulo Cesar Masiero[2]
{rtvb,masiero}@icmc.usp.br

Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo – Campus de São Carlos

Abstract

The GREN-Wizard, a tool to instantiate the GREN framework to particular systems in the business resource management domain, is presented in this paper. It was built based on a pattern language for the same domain, called GRN, which is used during the instantiation process. The GREN-Wizard automatically generates all the code needed to adapt the framework to the particular system, according to the input data supplied by the user. This data consists basically of information about the patterns of GRN applied to model the particular system, together with additional attributes included in the classes that compose each pattern.

Keywords: *Software reuse, tools, framework instantiation, pattern languages.*

1 Introduction

Software patterns and pattern languages aim at reuse in high abstraction levels. Software patterns try to capture the experience acquired during software development and synthesize it in a problem/solution form [8]. A pattern language is a structured collection of patterns that build on each other and can be systematically applied to produce complete applications. It represents the temporal sequence of decisions that led to the complete design of an application, so it becomes a method to guide the development process [6].

Object-oriented software frameworks (from now on called simply frameworks), allow the reuse of large software structures in a particular domain, which can be customized to specific applications. Families of similar but non-identical applications can be derived from a single framework. However, frameworks are often very complex to build, understand, and use. Framework instantiation, which consists of adapting it to specific application requirements, is complex and, most times, requires a complete understanding of its design and implementation details.

Pattern languages and frameworks can be used together to improve reuse even more [5, 4, 3]. Thus, the availability of a pattern language for a specific domain and its corresponding framework imply that new applications do not need to be built from scratch, because the framework offers the reusable implementations of each pattern of the pattern language. Therefore, the application development process may follow the language graph, from root to leaves, deciding on the use of each specific pattern and reusing its implementation offered by the framework.

In this paper we present the GREN-Wizard, a tool to instantiate the GREN framework to particular systems in the business resource management domain. The paper is organized as

[1]Financial support from FAPESP Process n. 98/13588-4.
[2]Financial support from FAPESP/CNPq.

follows. Section 2 presents the GRN pattern language and its associated framework (GREN). Section 3 introduces the GREN-Wizard and describes its main functionality. Section 4 presents the concluding remarks.

2 The GRN pattern language and the GREN framework

The tool presented in this work was built to take advantage of the relationship between a pattern language and its associated framework, as shown in Figure 1. The tool (GREN-Wizard) was built to support the automatic instantiation of applications using the GREN framework [1]. GREN, by its turn, was built based on the GRN pattern language [2]. Concrete applications can be generated by instantiating GREN manually or using the GREN-Wizard. In both cases, the GRN pattern language is first applied to model the system and then the framework is adapted according to the patterns used.

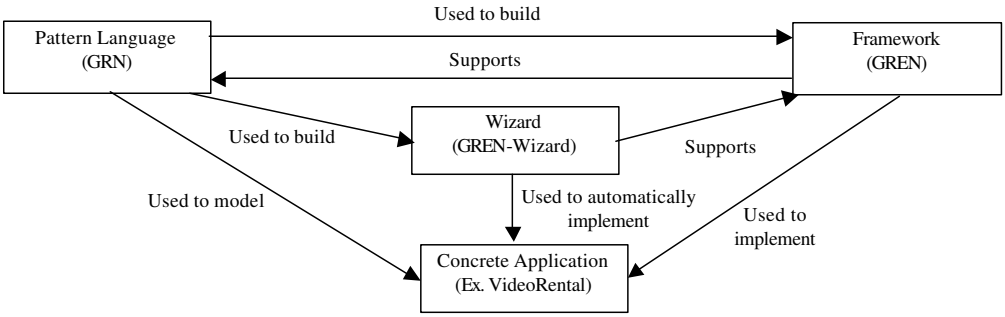

Figure 1: Relationship among pattern languages, frameworks, tools and applications

The GRN pattern language (*Gestão de Recursos de Negócios*, in Portuguese) was built based on the experience acquired during development of systems for business resource management. Business resources are assets or services managed by specific applications, as for example videotapes, products or physician time. Business resource management applications include those for rental, trade or maintenance of assets or services. The GRN pattern language has fifteen patterns that guide the developer during the analysis of systems for this domain. The first three patterns concern the identification, quantification and storage of the business resource. The next seven patterns deal with several types of management that can be done with these resources, as for example, rental, reservation, trade, quotation, and maintenance. The last five patterns treat details that are common to the seven types of transactions, as for example payment and commissions. All GRN patterns have a structure diagram that uses the UML notation. So, each pattern has participant classes, each of them with attributes, methods and operations. Besides, a pattern can have alternative solutions depending on the specific context in which it is applied. So, pattern variants are used to denote each possible solution to the same problem.

The GREN framework was developed to support the implementation of applications modeled using GRN. All the behavior provided by classes, relationships, attributes, methods, and operations of GRN patterns is available on GREN. Its implementation was done using the VisualWorks Smalltalk [7] and the MySQL DBMS [9] for object persistence. The first GREN version contains about 150 classes and 30k lines of code in Smalltalk. GREN instantiation consists of adapting its classes to particular requirements of concrete applications. This is done by creating subclasses inheriting from GREN abstract classes and overriding the necessary methods. As GREN has been built based on GRN, its documentation was done in such a way that, by

knowing which patterns and variants were applied, it is possible to know which classes need to be specialized and which methods need to be overridden.

3 The GREN-Wizard

The GREN-Wizard is a tool to support the GREN instantiation. It was designed so that framework users need only to know the GRN Pattern Language in order to obtain the Smalltalk code for their specific applications. So, the interaction with GREN-Wizard screens is similar to using the GRN Pattern Language. The user will be asked what patterns to use, what variants are more appropriate to the specific application, and which classes play each role in the pattern variant used. After, several choices will be offered to proceed with the application of other GRN patterns. The wizard is used in parallel with the pattern language application, i.e., the pattern language is used to model the system, producing an analysis model and a history of patterns and variants applied. This information is used to fill in the wizard screens and produce the code needed to adapt the framework to the particular application.

Figure 2 shows an example of a GREN-Wizard screen. New applications can be created by clicking in the "New" button (see #1 in Figure 2). The new application will appear in the box of applications (#2). The first pattern of GRN will appear in the box of the current pattern being applied (#3). Beginning with the first pattern of the list of applied patterns (produced during the analysis of the system, using the pattern language), the participating classes of each pattern are filled in. The pattern variant (#4) has to be chosen accordingly. When the pattern variant is changed, the participant classes are adapted according to it. It is important to make sure that all the participants that are really needed in an application are present in the variant chosen. After filling in all classes, the "Apply Pattern" button (#5) is used to save the current pattern in a list of applied patterns. The "Attributes" button (#8) can be used to change attribute names for the class or to add new attributes. Then, the next pattern to apply (#6) is selected and the "Go" button (#7) is used to make the wizard adapt its screen to the next pattern chosen. This procedure is followed until the last pattern is filled.

At any time during navigation through the GREN-Wizard, the "Patterns already applied" button can be used to see a current specification of the patterns that were applied. The texts of the several parts of the pattern language can also be seen, in case the framework user does not have it at hand (for example, button #9 shows an introductory text about the pattern language). Some applications require that GRN be applied in several iterations, probably for different resources being managed. To do that, the "Reinitialize for another Resource" button is used. Before being able to generate the concrete application classes, the current application specification has to be saved, using the "Save" or the "Save as" button. An optional activity is to specify which reports of the specific application will appear in the Main Window menu. The final steps are to generate the new classes and the associated code, and to automatically create the MySQL database.

The GREN-Wizard construction was eased by the fact that GREN had been built based on GRN. Figure 3 graphically illustrates how the GREN-Wizard works. A database (D1) is used to store data about the GRN pattern language, i.e., its constituent patterns, their classes and relationships. Through the GREN-Wizard graphical user interface (GUI), the user supplies information about the patterns used to model a specific application. This information is stored in the database (D2) and used by the GREN-Wizard code generator, together with information about the mapping between GRN and GREN (D3), to automatically produce the classes of the specific application.

The meta-model of Figure 4 shows part of the GREN-Wizard internal structure. Data about each pattern, its variants, classes, attributes and relationships are represented by classes of this

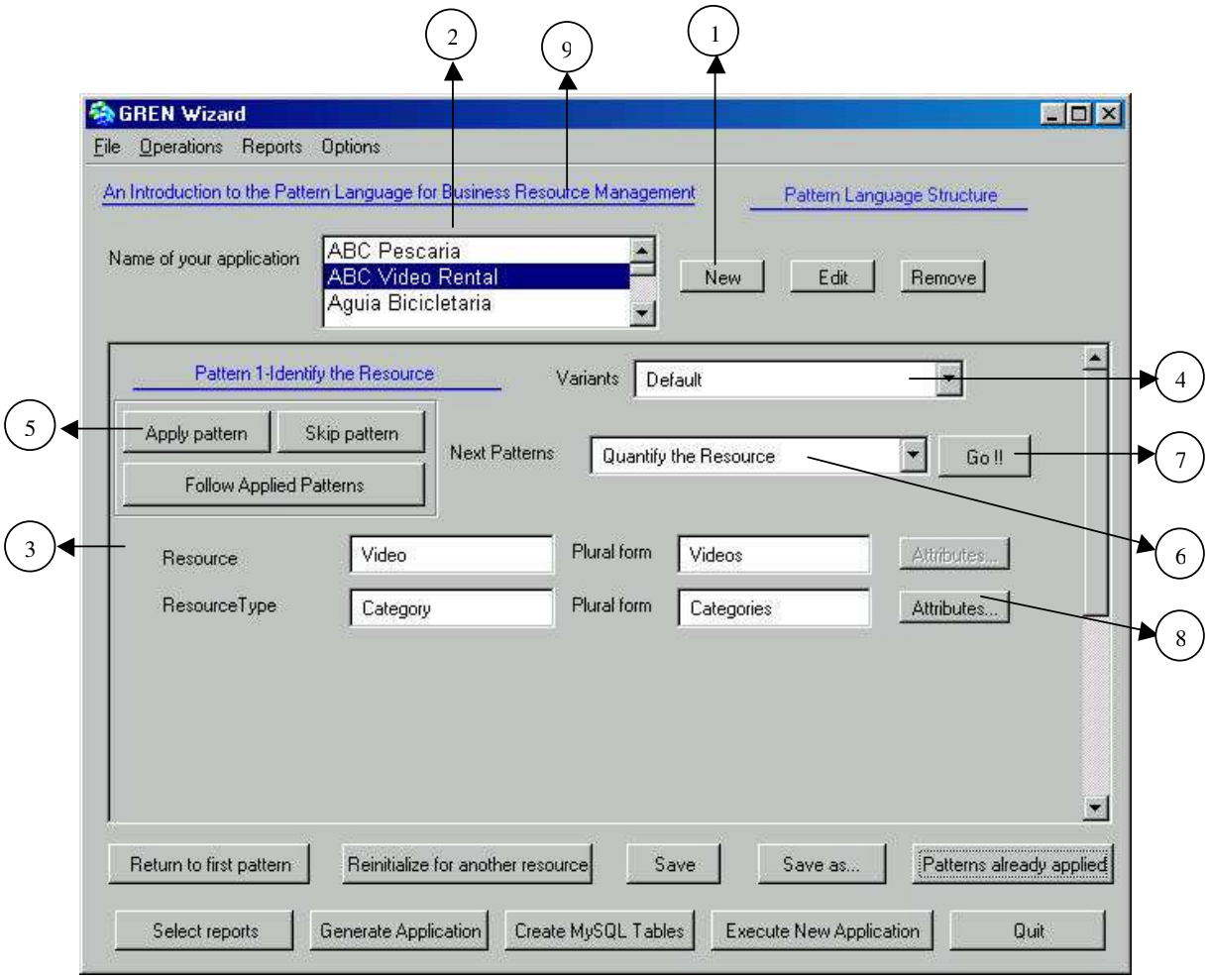

Figure 2: Example of the GREN-Wizard screen

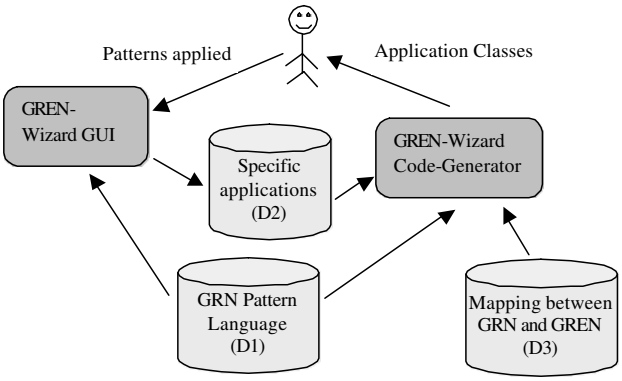

Figure 3: Illustration of the GREN-Wizard Strucutre

model, as well as some attributes that create a mapping from the pattern to the corresponding classes in the framework that implements it. For example, the attribute `superclassName` of class *PatternClass* maps the pattern class to the corresponding framework class. This allows

the wizard to identify which classes need to be specialized according to the patterns chosen by the user. Similarly, tables were created to hold information about methods that need to be overridden in the newly created classes.

The wizard allows the iterative application of the patterns, i.e., the application of one pattern more than once, and restricts the order of application of each pattern according to the relationship among the patterns established in the database. The definition of the specific database tables for the new system are also automatically generated and executed by the wizard. So, after running the wizard, the new application can be executed without additional programming.

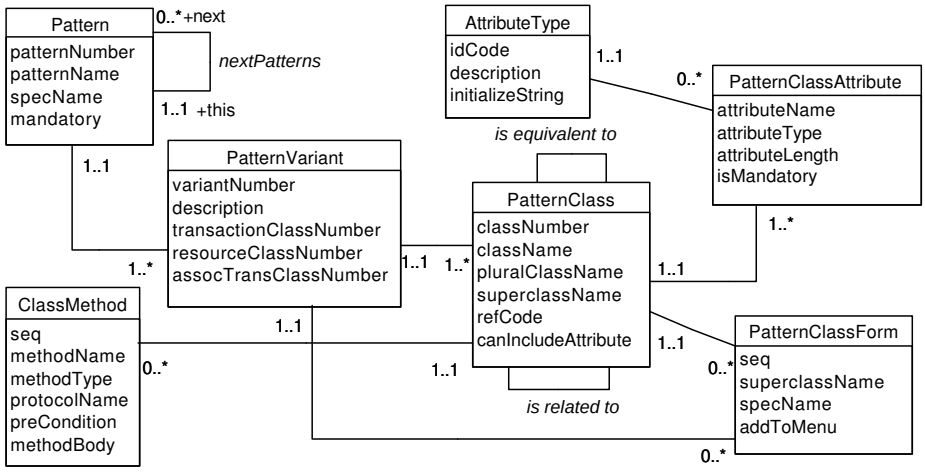

Figure 4: Part of the GREN-Wizard Meta-model

The database denoted by D2 in Figure 3 stores information about each GREN specific instantiation as, for example, which patterns/variants were applied (and in which order), the roles played by each class in the patterns, attributes added to pattern classes, etc. As the wizard is used to build many specific systems, this database grows and can be reused when similar applications need to be created. For example, having used the wizard to build a library system, if, in the future, a different library system needs to be created, the wizard can show the set of patterns to be used in the new system, as well as the attributes that can be added in this typical application. So, specific knowledge about the domain is stored by the GREN-wizard.

The same rationale used to build the GREN-Wizard can be used to develop tools for other frameworks that have an associated pattern language, since the idea is general. The fact that the wizard uses a database to hold the definition of each pattern eases its adaptation to other frameworks. Only a few attributes of the meta-model presented in Figure 4 are specific to GREN/GRN, as for example the `specName` attribute of class *Pattern* and the `protocolName` attribute of class *ClassMethod*, which are particular of Smalltalk implementation.

4 Concluding remarks

The tool here proposed intends to ease framework instantiation using pattern languages. Frameworks built using our approach have its architecture influenced by the pattern language, which eases the creation of a tool to automate its instantiation. Rather than knowing the framework implementation details, users basically need to know about the pattern language usage in order to instantiate the framework.

The GREN-wizard guides the user throughout the GRN pattern language, guaranteeing

that the correct paths are followed and making the consistency to check whether the applied patterns make sense. Thus, the framework user knows exactly where to begin and to finish the instantiation. Also, the instantiation is focused on the functionality required, with a clear notion of which requirements are attended by each pattern.

Several applications that were manually instantiated using the GREN white-box version were instantiated using the GREN-Wizard, both to test the wizard and to compare the results. The applications were: for a video rental system with 32 classes, for a sales system with 16 classes, for a car repair shop with 22 classes, and for a pothole repair system with 27 classes. The total number of lines generated by the GREN-Wizard was about 5000. The resulting applications could be executed properly, with the same functionality of the applications instantiated manually. The time required to develop these applications using the GREN-Wizard was approximately half an hour for each of them, while the same application, when instantiated manually using the white-box version of the framework, took approximately 10 hours. Other case studies are being conducted to demonstrate how much of the implementation can be achieved by using the wizard, and the difficulties of using the white-box version of the framework to implement the functionalities not automatically provided by the tool.

References

[1] R. T. V. Braga. GREN: A framework for business resource management. ICMC/USP – Sao Carlos, August 2001. Unpublished, Available on August, 2001 at: `http://www.icmc.sc.usp.br/~rtvb/GRENFramework.html`.

[2] R. T. V. Braga, F. S. R. Germano, and P. C. Masiero. A pattern language for business resource management. In *6th Pattern Languages of Programs Conference (PLoP'99)*, Monticello – IL, USA, 1999.

[3] R. T. V. Braga and P. C. Masiero. Frameworks construction and instantiation using pattern languages. In *Proceedings of the International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications, Foz do Iguazu-Brazil*, pages 305–310. ACIS, 2002.

[4] R. T. V. Braga and P. C. Masiero. A process for framework construction based on a pattern language. In *Proceedings of the 26th Annual International Computer Software and Applications Conference, IEEE Computer Society, to appear*, 2002.

[5] D. Brugali and G. Menga. Frameworks and pattern languages: an intriguing relationship. *ACM Computing Surveys*, 32(1):2–7, March 1999.

[6] D. Brugali, G. Menga, and A. Aarsten. *A Case Study for Flexible Manufacuring Systems*, pages 85–99. Domain-Specific Application Frameworks: Frameworks Experience by Industry, M. Fayad, R. Johnson, –John Willey and Sons, 2000.

[7] Cincom. Visualworks 5i.4 non-commercial, 2001. Available for download on September 25, 2001 at: `http://www.cincom.com`.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.

[9] MySQL. MySQL 3.23 version, 2001. Available for download on September 25, 2001 at: `http://www.mysql.com`.