SYSTEM DEVELOPMENT USING A PATTERN LANGUAGE-BASED TOOL

Rosana Braga, Paulo Masiero and Fernao Germano
ICMC - University of Sao Paulo, SCE
13560-970 Sao Carlos
Brazil

Keywords: System Development, Pattern Languages, Frameworks, Systems Engineering Methodologies

Abstract: Domain-specific pattern languages can be used to model applications, so that following particular paths in the pattern language lead to the complete design of particular systems. This paper shows how to use a pattern language-based analysis method and tool to help in the development of domain-specific systems, where the development is basically done at the analysis level. The requirements of the target system are matched against analysis patterns, so that the system is specified in terms of the patterns used to model it. The tool is fed with this information and uses it to instantiate a framework that was built based on the same pattern language. The result is the source-code for the target system, that can be used as a prototype, extended or improved to become the real system.

1 INTRODUCTION

Software reuse is a goal pursued since the beginning of the computer era, because reusing already tested artefacts leads to more productivity and quality in software development. Providing means of writing less code is an important topic of research, and in this sense, several approaches have emerged in the last decades, as for example object-oriented programming, domain analysis, software components, frameworks, and patterns. In particular, object oriented frameworks are composed of concrete and abstract classes that represent a family of systems for a specific domain, and they can be specialized to produce many different applications in that domain.

Software patterns document solutions to common problems found during software development, so that inexperienced developers can use these solutions when facing the same problems (Gamma et al., 1995; Aarsten et al., 2000). A pattern language is a structured collection of patterns that can be applied sequentially to obtain the entire architecture of a system (Coplien, 1998). A pattern language represents the temporal sequence of decisions that lead to the complete design of an application, so it becomes a method to guide the development process (Brugali & Menga, 1999). Pattern languages reflect experience in specific domains, covering all their main aspects. Consequently, particular systems of that domain can be specified in terms of the patterns applied to model them. A framework can be built based on a pattern language composed of analysis patterns that cover the desired functionality of a specific domain (XXX et. al., 2002). Such framework supports the implementation of applications modelled using the corresponding pattern language, so that the development can be focused on the analysis level, i.e., by knowing which patterns of the pattern language were used to model a specific application, it is possible to easily instantiate the framework to that application. A visual builder can aid this task, by automatically creating the source-code needed to produce specific applications.

In this work we postulate that the use of a pattern language-based tool, which automates the instantiation of a framework, built based on the same pattern language, substantially eases the development of domain-specific systems. They can be developed with no programming, focused only on the system functionality at the analysis level. This helps to shorten the gap between system requirements and implementation, as the patterns are

Braga R., Masiero P. and Germano F. (2004).
SYSTEM DEVELOPMENT USING A PATTERN LANGUAGE-BASED TOOL.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 155-162
DOI: 10.5220/0002639001550162

situated on the analysis level and can be easily mapped to the users requirements. At the same time, the framework construction, which was based on the pattern language, allows the mapping of the patterns into the implementation classes. The pattern language-based tool closes the cycle, allowing the user to inform the patterns used to model the system and automatically produce the application source-code that, together with the framework source-code, composes the final application.

A case study is used in this paper to illustrate the approach. It consists of the development of a pothole repair system, using GREN-Wizard (XXX & YYY, 2003), which is a tool to instantiate the GREN framework (XXX & YYY, 2002), based on GRN, a pattern language for business resource management (XXX et al., 1999). We use this particular pattern language and tool, but the approach is general and can be reused for other domains.

The focus of this paper is the use of the pattern language-based tool to obtain domain specific systems. Before using this tool, the target system is analyzed based on a pattern language, producing an analysis model marked with the patterns used to model it. This modelling is shown in Section 2. After automatically producing the source-code of the application, as described in Section 3, it has to be validated and may be extended to add functionalities not provided by the pattern language. This is described in Section 4. Section 5 discusses the proposed approach and related work. Finally, Section 6 presents the conclusions and future work.

2 MODELING WITH THE USE OF A PATTERN LANGUAGE

The first step of our approach is the system analysis based on a domain-specific pattern language. To begin with, it is considered that there is a domain-specific pattern language, composed of analysis patterns, which present solutions, in terms of class diagrams, to solve all the main problems found when modeling systems in that domain. Each pattern has a solution to a particular problem and the use of this pattern leads to a small class diagram representing part of the target system. After applying one pattern, the pattern language provides means of deciding about the next patterns to be applied.

For example, consider the GRN Pattern Language, for Business Resource Management (*Gestão de Recursos de Negócios,* in Portuguese), which was built based on practical experience acquired during development of systems for business resource management. We will use this pattern language to illustrate the process proposed in this paper. 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.

GRN has fifteen patterns (see Figure 1), that guide the developer during the analysis of systems of this domain. Its main patterns are RENT THE RESOURCE, TRADE THE RESOURCE, and MAINTAIN THE RESOURCE. Patterns are grouped according to their purpose: 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 business resources, as for example, rental, reservation, trade, quotation, and maintenance. The last five patterns treat details that are common to all the seven types of transactions, as for example payment and commissions.

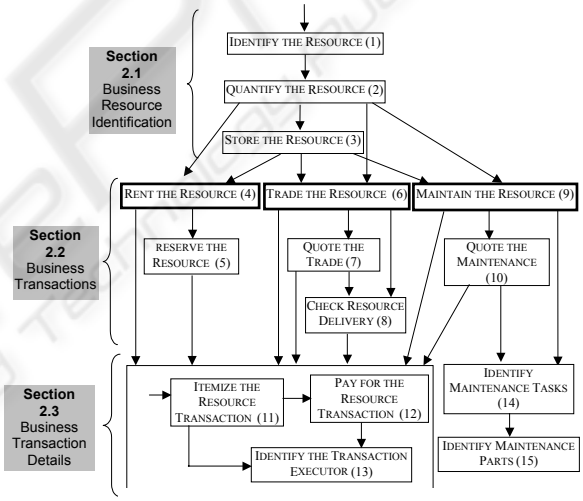


Figure 1: GRN Pattern Language

Figure 2 shows part of pattern 9, extracted from GRN. Observe that the pattern structure diagram uses the UML notation with some modifications. Special markers are included before input and output system operations, which are more than methods, as they are executed in response to system events that occur in the real world. A ? mark is used for input operations and a ! mark is used for output operations. A * mark before a method name means that its call message is sent to a collection of objects, instead of to a single instance, i.e., it will probably be implemented as a class method. 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. Pattern variants are used to denote each possible solution to the same

problem. A pattern can have optional participants, as for example, Source Party of Pattern 9, and this is explained in the "Participants" element. The "Following Patterns" element guides the user to the next patterns to be used.

Pattern 9: MAINTAIN THE RESOURCE

Context

Your application deals with resource maintenance or repair. You have already identified and quantified these resources, which are basically customer assets that present faults or need periodic maintenance. . .

Problem

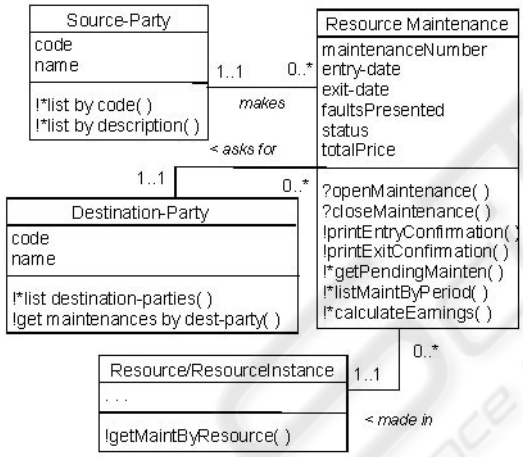
How do you manage resource maintenance performed by your application?

Forces

- Keeping maintenance records is important both to customers and to organizations that do maintenance

. . .

Structure



Participants

Resource Maintenance: represents all the details involved in maintaining a resource…

Source-Party: represents the organization department or branch responsible for doing the maintenance. This class is optional in this pattern because . . .

Example

. . .

Following patterns

Check now the other patterns in Section 2.3, which deal with other maintenance details. After that, check the convenience of using the QUOTE THE MAINTENANCE (10) and the IDENTIFY MAINTENANCE TASKS (14) patterns.

Figure 2: Example of a GRN Pattern

We consider that the software engineer is familiar with the pattern language, i.e., its domain, the patterns available, and how to apply them. It is also considered that a requirements artifact has been produced that describes the desired functionality for the system to be developed. So, this artifact has to be studied to allow the decision of whether the analysis pattern language and the corresponding framework can be used for the system development.For example, consider a Pothole Tracking Repair System (PHTRS), as the one defined by Pressman (2001). In this system, citizens can log onto a Web site and report the location and severity of potholes. As potholes are reported, they are logged within a "Public works department repair system" and are assigned an identifying number, stored by street address, size (on a scale of 1 to 10), location (middle, curb, etc.), district (determined from street address), and repair priority (determined from the size of the pothole). Work order data are associated with each pothole and include pothole location and size, repair crew identifying number, number of people on crew, equipment assigned, hours applied to repair, hole status (work in progress, repaired, not repaired), amount of filler material used, and cost of repair (computed from hours applied, number of people, material and equipment used).

After studying GRN, the software engineer easily recognizes that, in PHTRS, the pothole is the resource being managed and its repair is one of the transactions allowed to be made with resources, in particular it is a resource maintenance, so PHTRS can be modeled using GRN.

Once decided to go ahead, the pattern language has to be used to model the system. The pattern language has to be self-contained, in terms that it needs to have the necessary information to allow its application. In order to discipline the work, a class diagram can be sketched that shows the portion of the system that will use a certain pattern, using distinct colors or symbols to stress possible attributes, methods, or operations added to those of the pattern. This class diagram grows gradually as new patterns are applied. For each applicable pattern, a mark in the requirements document has to be made to indicate which requirements have been satisfied by each pattern.

UML stereotypes can be used to indicate the roles played by each class of the pattern. For each applicable pattern, its name has to be recorded, as well as its variant or sub-pattern, if it exists. If the pattern has an element "Following Patterns" or equivalent, then it is suggested which possible patterns to investigate after applying or not a certain pattern. This defines several possible paths to follow that should be recorded by the developer and

investigated individually. The requirements document then should be examined to detect requirements not satisfied or only partially satisfied by the pattern language. The class diagram then has to be complemented by the addition of new attributes, classes, relationships, methods, and operations, using a different color and registering in the requirements document the parts not covered by the pattern language.

A table containing the history of patterns and variants used could be prepared containing: the pattern applied; the variant or sub-pattern used ("default" should be marked if the pattern has been used as presented in the solution); the name of the class participating of the pattern; and the name of the application class that plays the role of the pattern class participant. This table is latter useful to supply information to be filled in the visual builder GUI forms. The result of this step is a class diagram complemented with information about the patterns used and the requirements not fulfilled by the pattern language. Alternatively, experienced developers can proceed directly to use the tool, based only on the analysis model.

In the PHTRS example, after applying several patterns, the system analysis model shown in Figure 3 is obtained, together with Table 1, which shows the history of patterns applied and roles played by each PHTRS class. The analysis model is marked with the patterns, using UML stereotypes, to ease the future interaction of the software developer with the visual builder. Notice, in Figure 3, that the P#N stereotypes are used to denote the patterns used and, at the same time, the roles played by a PHTRS class in each pattern. For example, Work Order plays the role of Resource Maintenance in patterns 9, 14, and 15, and the role of Transaction in pattern 13. These roles can also be seen in Table 1. In fact, this table contains the same data shown in the diagram, but organized in such a way to ease the next step, which consists of feeding the visual builder.

3 USING THE VISUAL BUILDER BASED ON THE SYSTEM MODEL MARKED WITH THE PATTERN LANGUAGE

The second step of our approach is the use of a visual builder or tool to implement the specific application. Based on the analysis model of the specific application, together with the log of the patterns and variants used, the tool is fed with the information needed to automatically generate the source code of the specific application classes.

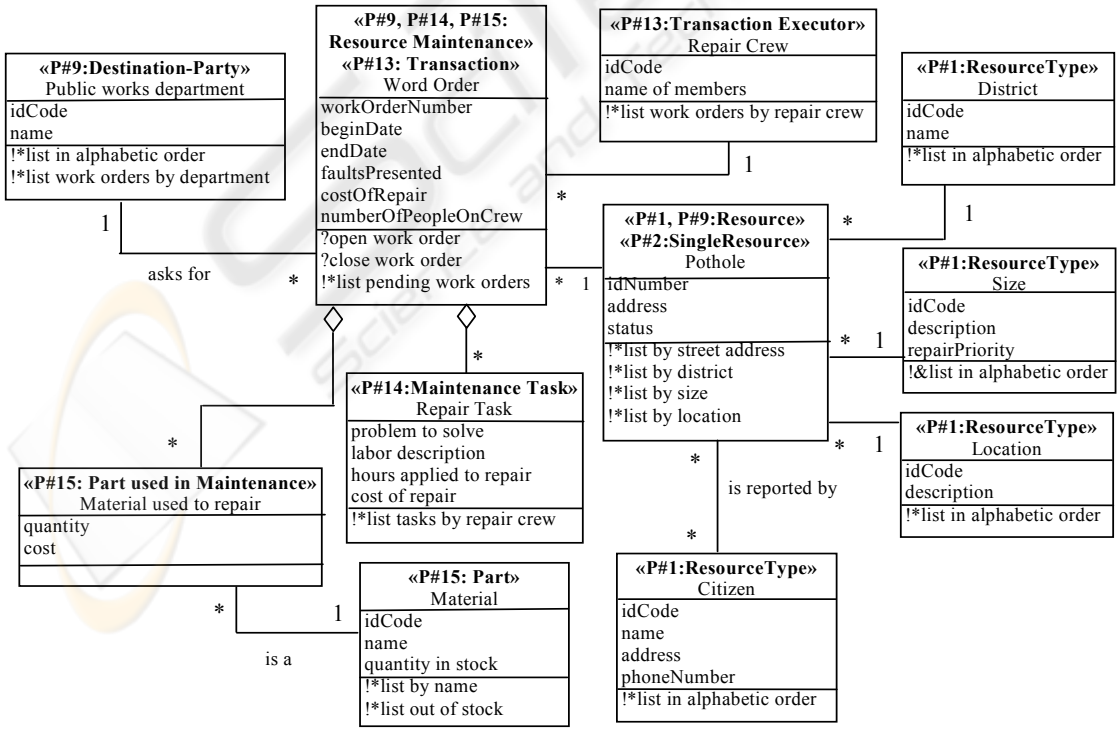


Figure 3: PHTRS Analysis Model

Table 1: History of patterns applied to model PHTRS

Pattern: 1 - Identify the Resource		
Variant	**Pattern Participant**	**Application Class**
Multiple types	Resource	Pothole
	Resource Type	District
	Resource Type	Size
	Resource Type	Location
	Resource Type	Citizen
Pattern: 2 - Quantify the Resource		
Single Resource	Resource	Pothole
Pattern: 9 - Maintain the Resource		
No source party	Resource	Pothole
	Resource Maintenance	Work Order
	Destination-Party	Public Works Department
Pattern: 13 - Identify the Transaction Executor		
No commission	Transaction Executor	Repair Crew
	Transaction	Work Order
Pattern: 14 - Identify Maintenance Tasks		
Transaction Executor instead of Task executor	Resource Maintenance	Work Order
	Maintenance Task	Repair Task
Pattern: 15 – Identify Maintenace Parts		
Default	Resource Maintenance	Work Order
	Part used in Maintenance	Material used to repair
	Part	Material

It is considered that a framework was built, based on the pattern language, to support the implementation of classes in the same domain of the pattern language. Also, there is a mapping between the patterns of the pattern language and the framework classes, i.e., for each participant class of a pattern, with its attributes and methods, it is possible to know the corresponding framework classes, attributes, and methods. This framework can be manually instantiated based on this mapping, but this task is time consuming and error prone. So, a visual builder can be built to help this task, with the following desirable features:

- It controls the sequence of application of the patterns, because a pattern language usually has restrictions about the order in which the patterns are applied, and the application of one pattern may require the application of one or more other patterns;
- It allows the future use of specifications when modeling similar systems, i.e., it has to log all the information about the patterns used to model a particular application, to ease other developments;
- It allows the application of one pattern more than once, because the same problem may occur several times during system development;
- It automatically creates the concrete application classes and database to persist objects, which can be done based on the mapping between patterns and framework classes.

Table 2 summarizes the five sub-steps involved in the use of a visual builder. The first sub-step may require iteration through the pattern language, as many real applications deal with more than one resource. For example, in a car repair shop, "car" is the resource being maintained and "part" is the resource being sold to the customer during the repair. Besides, "part" is the resource being bought from the supplier. So, there are two resources being managed in this example, the first related to a transaction (maintenance), and the other to two corresponding transactions (sale and purchase).

In the second sub-step, the user has to decide about which reports, among those offered by the pattern language, are part of the specific application requirements. For example, in PHTRS it is interesting to have a report of the pending work orders, i.e., work orders that are in progress. This corresponds to the system operation: "!*getPendingMainten()", which is part of the class diagram of Figure 2 (Resource Maintenance class). It is important to notice that basic system operations, such as inclusion or update of the work order, search of objects by idcode or description, etc., are automatically provided by GREN-Wizard, so they do not need to be chosen by the user.

Table 2: The process for using the visual builder

Step	Description
1 – Model definition	The model of the target application is informed to the visual builder, in terms of patterns applied.
2 – Choose reports	Output system operations, denoted in the pattern language with a ! mark, can be chosen to make part of the final application.
3 – Generate classes	Final application classes are generated, overriding the necessary methods and dealing with added attributes and classes.
4 – Generate database	Depending on the framework, the corresponding database is created to persist objects.
5 – Adapt the graphical user interface	Adjustments are made to the final application to adapt its graphical user interface according to the specific application.

Sub-steps 3 and 4 deal with code and tables generation, and depend on the particular framework. Sub-step 5 takes care of small adaptations done to the generated code, as for example changing labels and position of widgets in the GUI. More elaborated adaptations are part of the third step of our approach, shown in Section 4.

As an example, consider the GRN pattern language, introduced in Section 2. The GREN framework (XXX & YYY, 2002) 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 VisualWorks Smalltalk, and the MySQL DBMS for object persistence. The first GREN version contains about 150 classes and 30k lines of Smalltalk code.

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, several mapping tables can be consulted to determine which classes need to be specialized, and which methods need to be overridden, to obtain the concrete application.

GREN-Wizard is a visual builder to support 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 GUI forms is inspired on using GRN. In fact, they are used in parallel. The user will be asked which patterns to use, which variants are more appropriate to the specific application, and which classes play each role in the pattern variant used. For example, in Figure 4, Pattern 9 of GRN – Maintain the Resource – is being applied, so the user is feeding information about the resource maintenance. A specific variant has been selected, which allows the omission of a pattern participant (see Figure 2). After applying this pattern, several choices will be offered to proceed with the application of other GRN patterns.

Some characteristics of GREN-Wizard are: 1) storage of the pattern language meta-model, containing the patterns, their possible variants, the classes, methods, and attributes belonging to each pattern, the relationship among patterns, the possible sequence of application of the patterns, etc. For example, the fields of the form shown in Figure 4 are dynamically built from a database; 2) creation and storage of new applications, generated based on GRN, with information about the patterns used and the patterns application order. This information can be used to reengineer the application, or to build similar systems. Furthermore, reports about the history of patterns applied to model the application, can be shown; 3) addition of new attributes to the classes (other than the pattern attributes), which can be of a simple data type (for example, integer, string, float, etc.), obtained from a Table or discrete List, or a multi-valued type. The last three types make easier to include N to 1 and N to N relationships between classes; 4) reuse of attributes or classes from previous systems implemented with the builder. For example, if you have developed an application in which the attributes of the Customer class have been entered, you can reuse most of them when developing a Patient class; and 5) partial or total reuse of systems implemented with the builder, in the construction of similar systems.

Returning to our example, after PHTRS is fully specified in terms of GRN patterns, this information is saved using GREN-Wizard. Then, the reports that will be available in the final application are chosen, and the code generator is invoked to automatically create classes, methods, and the graphical user interface. Also, the MySQL tables are automatically created by GREN-Wizard, for objects persistence. The result of this step is a prototype for the PHTRS system that has to be tested and possibly extended.

4 PROTOTYPE VALIDATION AND EXTENSION

The third step of our approach is the adaptation of the source-code, to satisfy the requirements that are not covered by the pattern language and by the framework. This action is optional, as some applications can be fully generated in an automatic way. During the first step of our approach, when the target system was analyzed using the pattern language, the requirements document was annotated, to highlight all possible non-covered requirements. However, an elaborated test may help to find other more fine-grained requirements that should be fulfilled. So, a complete functional test has to be conducted, aiming at producing a list of adaptations to be done to the prototype, which will evolve into the real system.

Once decided to enhance the prototype with additional functionality, the framework technical documentation and code have to be understood, to allow the adaptations to be made. The result is the application specific code, that has to be submitted to new tests, before delivery to the final user.

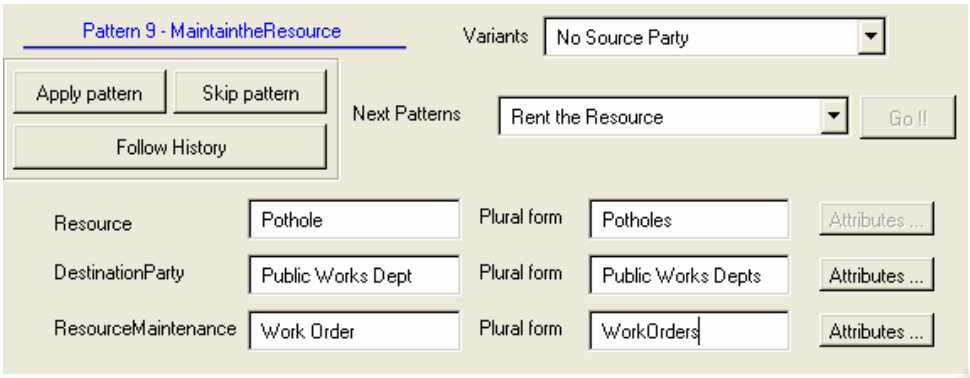


Figure 4: Example of the GREN-Wizard Graphical User Interface

It is important to notice that this step will depend exclusively on the coverage of the domain provided by the framework. A framework may cover less than half of the functionality of a particular domain, so a lot of work has to be done after instantiating it, or it can offer a 100% coverage of the functionality, implying in no programming at all.

In our PHTRS example, GREN-Wizard was able to cover all the functionality, so the system could be executed with no additional programming, except that we have improved the GUI with more meaningful labels and repositioned some widgets. The fact that GREN-Wizard allows the inclusion of typed attributes, as proposed by the Type-Object pattern (Johnson & Wolf, 1997), eases substantially the coverage of the domain, because classes that are not part of the patterns can be added to the model using these typed attributes. Nevertheless, if new functionality is needed by the system, the resulting source-code can be modified using the VisualWorks environment and, in this case, the GREN documentation needs to be studied by the software engineer, to determine which classes need to be modified or added. Adaptations should be evaluated to determine if they are worth to be implemented in the framework for future use, or if they are specific to the application and should be implemented only in the specific system.

5 RELATED WORK

The approach presented in this paper follows the patterns proposed by Roberts and Johnson (1998) for framework development, but differs in an important aspect: the use of a pattern language to guide the instantiation process. This makes the instantiation process easier, because it occurs in a higher semantic level than other framework instantiation processes. Another advantage, for the framework developer, is that it is possible to construct a visual builder without needing to gradually build a component library and developing black box objects.

Our approach is similar to the software product-line engineering approach, that aims at generating code for families of applications (Weiss and Lai, 1999). Both approaches generate code to be compiled and executed, but differ on how the code is generated. The product-line approach generates code from templates of code, whose place-holders are filled in by instructions supplied by the application developer, from a high level Application Modeling Language, while the visual builder instantiates a framework following the same procedure that would be followed by a human application developer, when instantiating the white-box version of the GREN framework. We think that this is only possible because the framework and the visual builder were developed based on a pattern language.

Similar work, concerning the relationship between pattern languages and frameworks, was done by Brugali et al. (2000), who have developed a framework for flexible manufacturing systems, based on a pattern language for the same domain. However, their pattern language is not an analysis pattern language, like the one presented in this work, and we do not know about the existence of a tool to automate the framework instantiation.

6 CONCLUDING REMARKS AND ONGOING WORK

System development can be enhanced with the aid of tools that accept information about the system in higher abstraction levels. Our approach uses domain specific pattern languages to achieve this goal, so that the reuse of object-oriented frameworks can be done without requiring the user to know the framework implementation details. Rather, they

basically need to know about the pattern language usage, in order to instantiate the framework. Thus, system development is focused on the functionality required, with a clear notion of which requirements are attended by each pattern.

The visual builder can also be used as an instrument to compose systems from patterns, choosing each pattern from its syntactic structure, instead of its semantic meaning in the pattern language domain, thus enlarging enormously the domain of applications that can be generated.

Several systems were developed by students using GREN-Wizard, with good results in productivity and requirements satisfaction. The analysis step, for these case studies, took in average 2 hours for medium applications such as: video rental with 32 classes, product sales with 16 classes, car repair shop with 22 classes, library with 24 classes, among others. The time required to develop these applications, using GREN-Wizard was approximately half an hour for each of them, while the same application, when manually instantiated using the white-box version of the framework, took approximately 10 hours. Notice that programming these applications from scratch would require several one-person-week work.

Other case studies are being conducted to evaluate the visual builder usability and the difficulties to implement the functionalities not provided by the framework. Some early results point that, a significant part of the non-attended functionality, can be used as feedback to improve the framework, while a minor part should be implemented only in the specific application.

As shown in this paper, the Visual Builder does not have a graphical interface yet. We are working now in developing an interface with the UML CASE tool ROSE, such that the patterns can be chosen from graphical templates, adapted according to the application semantic and exported to our tool using XML, for example.

GREN-Wizard is being used as a prototype creation tool, to support an agile process, named PARFAIT, for reengineering in the domain of business resource management (Cagnin et al. 2003).

Aspect based development (Kiczales, 1996) is being used within a Master's research to include non-functional requirements in applications instantiated from the framework/pattern language.

REFERENCES

Aarsten, A.; Brugali, D.; Menga, G. 2000. *A CIM Framework and Pattern Language*, in "FAYAD, M. E. & JOHNSON, R. E. (eds.). Domain-Specific Application Frameworks: Frameworks Experience by Industry, John Wiley & Sons.", p. 21-42.

XXX, X. X. X.; YYY, Y. Y. Y.; ZZZ, Z. Z. 1999 A Pattern Language for Business Resource Management. *Proceedings of the 6th Pattern Languages of Programs Conference* (PLoP'99), Monticello-IL, USA, v.7, p. 1-34.

XXX, X. X. X.; YYY, Y.Y., 2002. A Process for Framework Construction Based on a Pattern Language. In: Proceedings of the 26th Annual International Computer Software and Applications Conference, Oxford, *IEEE Computer Society*, 2002. p. 615-620

XXX, X. X. X.; YYY, Y.Y., 2003. Building a Wizard for Framework Instantiation Based on a Pattern Language. In: 9th International Conference on Object-Oriented Information Systems, Genebra, Suiça. *Lecture Notes on Computer Science*, LNCS 2817, Springer, p. 95-106.

Brugali, D.; Menga, G.; Aarsten, A. 2000. A Case Study for Flexible Manufacturing Systems, in in "Fayad, M. E. & Johnson, R. E. (eds.). Domain-Specific Application Frameworks: Frameworks Experience by Industry, John Wiley & Sons.", p. 85-99.

Brugali, D. & Menga, G. (1999). Frameworks and Pattern Languages: an Intriguing Relationship. In *ACM Computing Surveys*, march 1999.

Cagnin, M.I.; Maldonado, J.C.; Penteado, R.; Germano, F. 2003. PARFAIT: Towards a Framework-based Agile Reengineering Process. In: Agile Development Conference (ADC'2003), IEEE Proceedings, p. 22-31.

Coplien, J.O. 1998. Software Design Patterns: Common Questions and Answers, in *Linda Rising (editor) (1998) The Patterns Handbook: Techniques, Strategies, and Applications*, Cambridge University Press, New York, p. 311-320.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. (1995). *Design Patterns – Elements of Reusable Object-Oriented Software*. Reading-MA, Addison-Wesley.

Johnson, R. & Woolf, B. 1997. Type Object, chapter 4, In "Martin et al. et al. (1998), Pattern Languages of Program Design 3, Reading-MA, Addison-Wesley", p. 47-65.

Kiczales, G. 1996. Aspect-oriented programming. ACM Computing Surveys, v. 28, n. 4es, p. 154.

Pressman, R. S. Software Engineering – A Practitioner's Approach, 2001. 5th edition. McGraw Hill.

Roberts, D. & Johnson, R. E. (1998). Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks, in "Martin et al. (1998), Pattern Languages of Program Design 3, Reading-MA, Addison-Wesley", p. 471-486.

Weiss, D. M. & Lai, C. R. R. 1999. Software product-line engineering. Addison-Wesley.