Identification of Framework Hot Spots Using Pattern Languages

Rosana T. Vaccare Braga[1]
Paulo Cesar Masiero[2]
ICMC-Universidade de São Paulo
{rtvb, masiero}@icmc.sc.usp.br

Abstract

One of the major factors that brings complexity to framework development is the identification of its hot spots, i.e., the framework parts that must be kept flexible as they are specific of individual systems. In this paper we show that pattern languages can be important sources for framework hot spots identification and, consequently, can be used for framework construction. We define the types of hot spots that are identifiable from information presented in the elements of each pattern of the pattern language. We propose also a process for hot spots identification and present a case study for identifying the hot spots of a framework built based on a pattern language for business resource management.

1. Introduction

Software reuse is a goal that was set almost simultaneously with software engineering. Structured programming, followed by object-oriented programming and domain analysis were achievements obtained a long time ago aiming at the enhancement of software reuse. Object-oriented software frameworks[*] have emerged in the same context. They allow the reuse of large structures in a particular domain, which can be customized to specific applications in the domain. Families of similar but non-identical applications can be derived from a single framework.

Software patterns and pattern languages have also emerged aiming at reuse, but in higher abstraction levels. Software patterns try to capture the experience acquired during software development and synthesize it in a problem/solution form [1, 15]. A pattern language is a structured collection of patterns that build on each other to transform needs and constraints into an architecture [14]. Pattern languages represent the temporal sequence of decisions that lead to the complete design of an application, so it becomes a method to guide the development process [10].

Although the first frameworks were concerned with basic domains, as for example user interface, presently there is a great interest in developing application frameworks, i.e., frameworks for specific application domains, as business, engineering, medicine and insurance. Particularly, the business environment is becoming more dynamic with the enhancement of worldwide competition and market changes [11], demanding new technologies to decrease the applications development effort.

However, frameworks are often very complex to build, understand and use. There are a few methods for framework development, of which we can mention the hot-spot driven framework development [23, 24] and the framework design by systematic generalization [26, 27]. In both there is a step in which the framework hot spots must be identified, and this involves deep knowledge about the domain for which the framework is being built.

[1] Apoio financeiro da FAPESP – processo nº 98/13588-4
[2] Apoio financeiro do CNPq e FAPESP
[*] From now on we will use the term framework with the same meaning as object-oriented software framework.

Framework hot spots are the parts that have to be kept flexible, as they are specific of individual systems. They are designed to be generic and need to be adapted according to the requirements of each application that can be instantiated using the framework. Generally they are discovered first by domain analysis and then by successive framework refinements. However, each new discovery may imply in the need to redesign part of the framework, which makes development more complex. The best approach is to know beforehand which are the framework hot spots, in order to minimize the number of iterations needed for its construction.

Pattern languages reflect experience in specific domains, covering all their main aspects. Consequently, they have built-in information about the points that differ from one application to another. According to Brugali and Menga there is an intriguing relationship between pattern languages and frameworks [9]. Both are conceived for a specific domain, solving most of the problems that are common to applications in that domain. Pattern languages may help to generate frameworks, as they contain the main abstractions found in an application domain. These abstractions can originate the framework high-level components. So, 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. Thus, 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 work we show that pattern languages can be important sources for framework hot spots identification and, consequently, can be used as a basis for framework construction. We define the types of hot spots identifiable from information presented in the pattern language. We also present a case study for identifying the hot spots of a framework built based on a pattern language for business resource management [5, 7].

The paper is organized in five sections. In section 2 we present the related work concerning framework hot spots identification. In section 3 we state some guidelines for identifying hot spots from pattern languages. In section 4 we show a case study and in section 5 we present the conclusions and future work.

2. Related Work

Pree proposes a process for framework construction [24]. In it, an object model for a specific application is initially defined, followed by a construction cycle that is repeated through successive framework refinements. Several steps are included in this cycle. First, the hot spots are identified and documented by hot spot cards. Then the hot spots are designed and implemented, and the framework is tested to assess whether the hot spots satisfy the domain requirements. New hot spots may be found in this step and the cycle is repeated. At the end of each iteration, the framework is evaluated to determine if it can be released for use or has to be modified.

Schmid states that frameworks are constructed by systematic generalization based on the class model of a fixed application [26, 27]. A high-level hot spots analysis is initially done, aiming at establishing the main aspects of the system that need to be kept flexible. Then each hot spot is analyzed in detail producing its specification. The next step is to make the high level design of each hot spot, generating several hot spot subsystems. The final step is to transform the fixed application class model into the framework class model, replacing groups of classes of the original model by the corresponding hot spots subsystem.

Froehlich and others document the framework hot spots, called hooks, using descriptions that ease the framework instantiation for particular applications [18]. The hooks are identified by the framework builder, considered to have more knowledge about the framework. This knowledge is stored in the hooks description and can be used by the application developer, who will know exactly what needs to be completed or extended in the framework and which choices need to be done to develop specific applications.

Roberts and Johnson present a pattern language for developing object-oriented frameworks, called "Evolving Frameworks" [25]. The patterns suggest that three concrete applications be developed first and be gradually generalized to produce a white box framework. Then it goes through several iterations to become more and more black box, making use of component libraries, pluggable objects, fine-grained objects, and finally of a visual builder and language tools. The authors suggest hot spots to be encapsulated in objects, so that variation is achieved by composing the desired objects rather than by creating subclasses and writing methods.

Bosch and others present a simple model for framework development with six phases [3]. The first phase deals with the domain analysis, which is necessary to describe the domain covered by the framework and to capture its main requirements and concepts, resulting in a domain analysis model. In the second phase the framework architectural design is created; in the third phase this design is refined; in the fourth phase the framework is implemented using a specific programming language. In the fifth phase the framework is tested to evaluate both its functionality and usability. Finally, in the sixth phase the framework is documented, usually with a user guide and design documentation.

We observe that the processes suggested by Pree, Schmid and Robert & Johnson begin with particular application models, including the desired flexibility later. In Froehlich's approach the knowledge about the hot spots comes from the framework developer, who needs to have acquired it through practical experience or domain analysis. In Bosch's approach the domain analysis model is obtained in the beginning, which makes the framework hot spots more foreseeable. In this work we propose the use of a pattern language to guide the framework development, particularly to identify its hot spots. This task is part of a larger work in which the pattern language is also used to help in the design of the framework and in the instantiation of applications from the framework.

3. Hot Spots Identification from Pattern Languages

There are several types of hot spots in a framework, which must be adapted to produce specific applications. According to Froehlich and others [18], a hot spot may be adapted in five different ways, summarized in Table 1. To perform such adaptation, the framework user may have three different support levels: 1) option level (there are pre-built components to be chosen); 2) support pattern level (parameters or values are supplied for a hot spot, following a behavior pattern); and 3) open-ended level (the requirements must be implemented by the user, without framework support). In a more general way, the adaptation of a framework for a specific application involves the inclusion of new classes, methods and attributes; the removal of default classes that are not desired in a particular application; and the adaptation of algorithms that calculate specific attributes of the application.

The existence of a pattern language for a particular domain can greatly help in the hot spots identification. The following subsections give more detail about the several sources of hot spots found in pattern languages.

Table 1 – Types of adaptation for a hot spot [18]

Type	Description
1	A feature that exists in the framework, but is not part of the default implementation, is enabled
2	A feature that is part of the default implementation, but is not desired, is disabled
3	A feature is substituted or overridden by another
4	The existing behavior is augmented
5	A feature or service is added

3.1. Pattern Language Graph, Context and Related Patterns

By analyzing the structure of a pattern language, we observe that it has several inter-related patterns. This relationship is generally shown through a graph that represents the interaction among patterns and the sequence in which they are applied. A simple analysis of which patterns are of mandatory use and which are of optional use, indicates several framework hot spots. For example, both the pattern language for improving the capacity of reactive systems [22] and the pattern language for business resource management [5, 7] use a graph to illustrate the language structure. By analyzing this graph we can have an idea of which patterns are mandatory or optional and, so, we can identify several hot spots of the framework to be built. For example, in the pattern language for business resource management the "Reserve the Resource" pattern is optional. This indicates the need of a hot spot in the framework to handle this feature. This can be confirmed by analyzing the pattern context, which presents the scenario for the pattern usage.

If the pattern language has no corresponding graph, information necessary to know whether each pattern is optional or not, can be found mainly in sections "Context", "Following patterns", and "Related patterns". The context is important, as it gives indications of the desirable features for the pattern usage. Adding to this the knowledge about the application domain, we can identify applications that do not fit in the context and that, consequently, do not use the pattern. Sections "related patterns" and "following patterns" show other patterns that are related to the current pattern, helping to identify other alternative patterns and, consequently, indicate the pattern optionality. The pattern language Accounts and Transactions [20] – although written in the Alexander form and, consequently, without a "related patterns" section – has a paragraph at the end of the "solution" section pointing to the related patterns, where it is clear that the user is directed to alternative patterns according to the application characteristics.

In general the hot spots found in these sections belong to types 1 and 2 of Table 1 and can be adapted through option, which is the simplest type of adaptation for the user. If the framework developer wants the pattern application to be the default behavior of the framework, then type 2 is used, otherwise, type 1 is chosen.

3.2. Variants and Sub-patterns

The "variants" or "variations" section, commonly found in the patterns of a pattern language, is one of the richest sources of hot spots. It contains alternative solutions for the problem solved by the pattern and defines variable aspects that must be available for the framework user to instantiate it. These variants may also be documented in pattern languages in the form of sub-patterns or through several solutions. For example, the "Buffalo Mountain" pattern, which is part of the generative development-process pattern language [13], has three sub-patterns with alternative solutions for the same problem. Sub-patterns have the same semantic meaning of variants and, so, are also important sources of hot spots. The pattern "Representing inheritance in a relational database", which is contained in the pattern language for Object-RDBMS Integration [8], has two alternative solutions for the same problem, giving rise to a framework hot spot.

Similarly to the hot spots found in section 3.1, the hot spots found in this section are mostly of types 1 and 2 of Table 1 and can be adapted through option. All the alternative solutions can be implemented and one of them is made default. During instantiation, the framework user chooses the desired solution.

3.3. Participants and Collaborations

The "participants" and "collaborations" sections, which are present in patterns that follow the GoF format [15], can indicate some of the framework hot spots, as they describe the participants of the pattern and their collaboration, giving alternatives to use or not some of the participants. When a participant is optional, there is a description of how the pattern works without it. For example, in the pattern language for business resource management [5], "Source-Party" is an optional participant in the "Rent the Resource" pattern, because small organizations do not have branches or departments to be managed. When a participant has alternatives, several classes are offered and one of them must be chosen to act as the participant, according to design decisions. For example, in the pattern language for Object-RDBMS Integration [8] there is a section named "Discussion" in which variations of the solution are presented to discuss the participants and their collaborations.

It is also possible to identify, in these sections, some hot spots resulting from propagation of previous design decisions. This means that some patterns may have their participants or relationships modified according to the pattern variants already applied. For example, in the pattern language for business resource management [5], the choice done when applying the "Quantify the Resource" pattern implies in several additions to the participants of patterns applied subsequently. In general, the hot spots found in these sections are of type 2 and 3 of Table 1 and can be adapted through option or support pattern.

3.4. Implementation

The "implementation" section, also present in patterns that follow the GoF format, contains suggestions of alternative implementations of the proposed solution, so that according to the restrictions imposed by each particular application, different implementations can be chosen. Thus, this section identifies another type of hot spot. It must be observed that the framework developer often makes implementation choices that limit the possible implementations to one or two solutions. So, it is common for the framework not to cover all the possibilities presented in the "implementation" section. If framework users want to take advantage of such possibilities, they must use the open-ended mode. In this section we can identify hot spots of types 1 to 4 of Table 1, so that all the desired flexibility can be implemented.

3.5. Structure

Another source of hot spots is the "structure" section, which contains a diagrammatic representation of the pattern classes and their relationships. A detailed analysis of this section can help to identify alternative behaviors that may be desired for the system operations, often not described in the "participants" section. Thus, new hot spots can be defined to allow, for example, new attributes or methods for the classes and alternative algorithms for computing attributes. This consists mainly of hot spots of types 4 and 5 of Table 1.

3.6. Types of hot spots identifiable from pattern languages

Table 2 summarizes the types of hot spots that can be identified from a pattern language. An identification code is assigned to each of them to be used in section 4. The adaptation type shows what has to be done to obtain an application from the framework.

Table 2 – Hot spots identifiable from pattern languages

idCode	Hot spot Description	Adaptation type	Main sources in the pattern language
PATTERN_ OPTION	Optional pattern	Several classes and relationships are disabled	• Language graph • Following patterns • Related patterns
PARTIC_ OPTION	Optional participant	One class and its relationships with other classes are disabled or enabled	• Participants
PARTIC_ CHOICE	Choice of participants	One or more participants must be chosen according to the system requirements	• Participants • Structure • Variants
RELATIONSHIP	Change of Relationship	One or more relationships must be changed according to the system requirements	• Participants
BEHAVIOUR	Change of Behavior	One or more algorithms must be changed according to the system requirements	• Participants • Structure
PROPAGATION	Propagation effect of application of another pattern	Some participants may have changes in attributes or methods according to other patterns already applied	• Participants • Structure

Comparing the adaptation type proposed by Froehlich and others (Table 1) with the adaptation type provided after identifying the hot spots through the pattern language (Table 2), we can see that the second allows a higher abstraction level than the first. This implies in a better understanding of the hot spots by the application developer, as the hot spots are more straightly linked to the system requirements. Furthermore, each of the hot spots is tied to a specific pattern of the pattern language. Thus, if the application instantiation follows the pattern language then the hot spots adaptation can be done more easily, because only those hot spots corresponding to the actually applied patterns must be considered.

3.7. Guidelines for hot spots identification

Based on the types of hot spots defined in Table 2 and on the information contained in a pattern language presented in sections 3.1 through 3.5, a generic process is proposed below to help the framework developer to identify the hot spots using a pattern language.

a. Initially analyze the pattern language graph, if there is one. Look for paths that skip one or more patterns. If no graph is available, try to look at the context of each pattern and also related patterns or next patterns to be applied, as explained in section 3.1.
b. Analyze each pattern of the pattern language, following the explanations supplied in sections 3.2 to 3.5 to identify possible hot spots.
c. For each hot spot identified in steps (a) and (b) do the following:

c1. Include it in the hot spots table, assigning it a number, a name and a brief description of the desired flexibility.
c2. Inform the hot spot type, according to what is needed to adapt the framework for a specific application (Table 2).
c3. Associate the hot spot with its source in the pattern language and the pattern number.

d. After having finished the analysis of the pattern language, analyze the table created in step (c) to:
d1. Refine the specification of each hot spot to include enough information for its subsequent design and implementation.
d2. Identify other hot spots that are not explicit in the pattern language, but should be included because they would bring more flexibility to the framework.
d3. Use the information about these new hot spots as feedback to improve the pattern language.

e. Consider now other non-functional aspects of the application that might originate new hot spots, which include portability, usability, security and reliability. Also consider design and implementation issues that would bring more flexibility.

Some considerations need to be done about steps *d2* and *e*, as they are not trivial activities. Knowledge about the domain is essential to perform activity *d2*, but some guidelines can give the framework developer indications of other sources of hot spots. For example, looking at class attributes in the patterns of the pattern language, some questions should be answered, like: "is this a computed attribute?" If so, "is it possible to have several types of algorithms to compute it?" If the answer is affirmative, a new hot spot has been found. Looking at class relationships, another source of hot spots is to argue the cardinality of the relationships. If it is possible to find applications where the cardinality would be different from the cardinality proposed in the pattern, then a variant of the pattern exists and, consequently, a new hot spot. It is also desirable to look for similar hot spots in the table, because sometimes new hot spots can be derived by analogy. Examples of some of these cases are supplied in section 4.

Regarding activity *e*, it is necessary to balance performance versus flexibility when considering non-functional requirements and design or implementation issues, because including several alternatives in the framework would make it more flexible but degrade system performance. For example, if we consider database portability, the choice of using a relational database or an object-oriented one may derive a hot spot to be set by the framework user. Another example is the graphical user interface, which could have two or more implementations (one for traditional applications, another for virtual applications, etc.) so that the framework user could choose one of them. Notice that this type of flexibility could be achieved by implementing several versions of the framework, which would cause less impact on system performance. An example of a design/implementation issue that could generate a hot spot is a web-based education framework, where the course selection mechanism can vary [16]. For example, the entire list of available courses or just the ones related to the student major could be shown.

4. Case Study

This section presents a case study performed to evaluate the proposed process. Section 4.1 summarizes the pattern language for business resource management, called GRN [5, 7], which was used to develop a framework for the same domain, called GREN [6]. Section 4.2

describes the application of the process proposed in section 3.7 for the identification of the GREN framework hot spots.

4.1 Pattern Language features

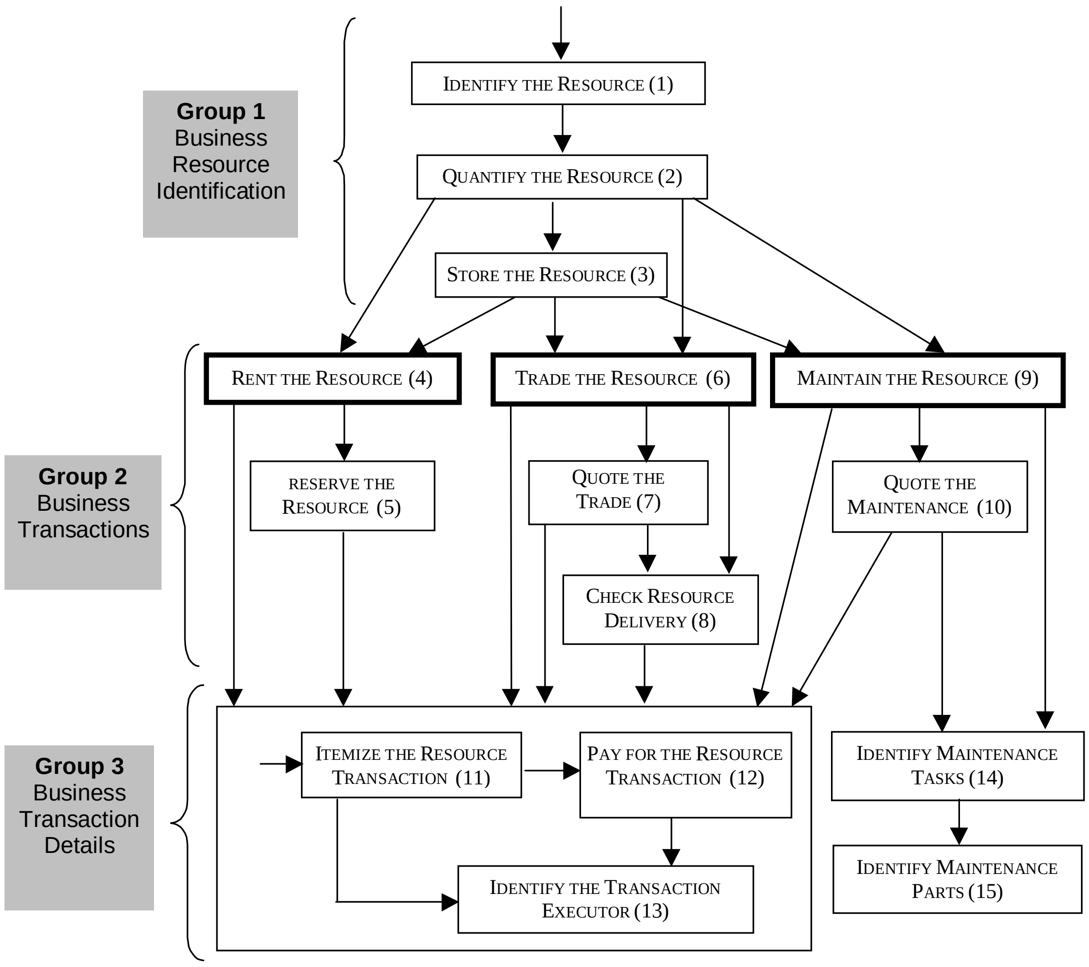
The Pattern Language for Business Resource Management (*Gestão de Recursos de Negócios*, or *GRN*, in Portuguese) [5, 7], originated from a family of three patterns for the same domain [4], whose patterns were split into smaller patterns to form the pattern language. It is composed of fifteen analysis patterns, some of which are specific usages or extensions of more generic patterns proposed in the literature [2, 12, 17, 21]. It is the result of an evolution of more than ten years of systems development practice of the first author for medium and small business in this domain. It was conceived to help software engineers in the development of applications concerned with business resource management. This includes applications where it is necessary to log transactions of business resource rental, trade or maintenance. By transaction we mean the same as Coad et al.: "a significant event to be remembered, i.e., an event that the system must remember through time" [12]. Resource rental focuses primarily on the satisfaction of a certain temporary need of a product or service like a videotape or a physician time. Resource trade focuses on the transference of property of a product, as for example a product sale or auction. Resource maintenance focuses on the maintenance of a certain product, using labor and parts to perform it, as in an electric appliance repair shop.

Figure 1 shows the dependencies among the patterns and the order in which they are generally applied. These dependencies are also presented, and eventually complemented, inside each specific pattern. The main patterns in the language are RENT THE RESOURCE, TRADE THE RESOURCE, and MAINTAIN THE RESOURCE, indicated by a thicker line. Their use is not mutually exclusive and, in fact, there are applications in which they can fit together. MAINTAIN THE RESOURCE may use RENT THE RESOURCE and TRADE THE RESOURCE, as in a car repair shop system, in which parts are traded and labor is rented. The patterns are grouped according to their purpose, as illustrated in Figure 1: group 1 patterns are basically concerned with the identification, quantification and storage of the business resources; group 2 patterns deal with the business transactions performed by the system; and group 3 patterns take care of details associated to most business transactions.

The GREN framework construction based on the GRN pattern language is being conducted in two phases. In the first phase a white box version of the framework is being built. The framework hot spots were identified and are being implemented. Besides the classes that refer to each pattern of the pattern language, other classes are necessary to deal with more general aspects (as for example object persistence, graphical user interface, security, etc.). In the second phase a wizard is being built, also based on the pattern language, to help instantiating the framework to specific applications.

4.2 Application of the hot spots identification process

The process outlined in section 3.7 was applied to identify the hot spots of the GREN framework. Most of the hot spots (88,9 %) were found based in the GRN pattern language (steps *a* and *b* of the process) and only 11,1 % were identified by other ways (step *d* of the process, in this case). Table 3 summarizes the types of hot spots found. We observe that most of them is of type PATTERN_OPTION, since the pattern language has many optional patterns and this results directly in hot spots, as explained in section 3.1. The appendix shows the mapping between each hot spot identified and the corresponding sources in the pattern language (when applicable) for the 36 hot spots found.



Group 1 Business Resource Identification
Group 2 Business Transactions
Group 3 Business Transaction Details
IDENTIFY THE RESOURCE (1)
QUANTIFY THE RESOURCE (2)
STORE THE RESOURCE (3)
RENT THE RESOURCE (4)
TRADE THE RESOURCE (6)
MAINTAIN THE RESOURCE (9)
RESERVE THE RESOURCE (5)
QUOTE THE TRADE (7)
QUOTE THE MAINTENANCE (10)
CHECK RESOURCE DELIVERY (8)
ITEMIZE THE RESOURCE TRANSACTION (11)
PAY FOR THE RESOURCE TRANSACTION (12)
IDENTIFY THE TRANSACTION EXECUTOR (13)
IDENTIFY MAINTENANCE TASKS (14)
IDENTIFY MAINTENANCE PARTS (15)

Figure 1 – "GRN Pattern Language" structure

To illustrate the hot spots identification based on the pattern language, we will use one pattern of the GRN pattern language, shown in Figure 2. It refers to the "Trade the Resource" pattern, extracted from the extended version of the pattern language [7]. The four hot spots identified from this pattern correspond to numbers 13 to 16 of the appendix. Hot spot 13, called "Trade the Resource", provides the flexibility to make optional the application of this pattern, because the pattern language can also be used for rental or maintenance applications, and so it may be desired not to apply this pattern. This is a PATTERN_OPTION hot spot, as the whole pattern is optional. It was identified during step *a* of the process of section 3.7. Analyzing the pattern language graph (see Figure 1), we notice that there are paths that do not include the Trade the Resource pattern, which implies that this pattern is optional. Also, observing the pattern context (see item 6.1 of Figure 2), we notice that applications that do not deal with resource trade do not fit in the proper context for the pattern usage and, thus, should not use it.

Hot spots 14 to 16 were found during step *b* of the section 3.7 process. Hot spot 14, called "Source-party existence", gives small organizations the possibility of having a simpler system, in which branches or departments are not considered. This hot spot was identified observing the "participants" section of the pattern (see item 6.5 of Figure 2: notice that "Source-party" is an optional participant of this pattern). It is a PARTIC_OPTION hot spot, as its purpose is to make source-party optional. Hot spot 15 ("Destination-party existence") is similar to number 14 and was identified in the same section.

Table 3 – Summary of the types of hot spots identified

Type of hot spot	Quantity found				Percentage
	Step a	Step b	Step d	Step e	
PATTERN_OPTION	14				38,9 %
PARTIC_OPTION		7			19,4 %
PARTIC_CHOICE		3			8,3 %
RELATIONSHIP		1			2,7 %
BEHAVIOUR		2	3		13,9 %
PROPAGATION		5	1		16,7 %
Total	**36**				

Hot spot 16 ("Traded quantity entry") concerns the inclusion of a new attribute in the trade due to the previous use of another pattern of the language and, so, is a PROPAGATION hot spot. The knowledge about the propagation caused by the application of certain patterns is embedded in the pattern language, as can be verified in the participant "Resource Trade" (see section 6.5 of Figure 2). The pattern language states that an attribute "quantity" is included in this class when the "Measurable Resource" sub-pattern has been applied earlier.

Another interesting result of the case study was the identification of four new hot spots during step *d2* of the process proposed in section 3.7. For example, hot spot 12 of the appendix was not identified from the pattern language, but found through inspection of similar hot spots in the table. An analysis of the table was done to check the propagation effect of applying the "Quantify the Resource" pattern. This pattern has four alternative solutions, presented as sub-patterns. One of them, the "Measurable Resource" sub-pattern, is used when the resource is dealt with in quantities, so the "quantity" attribute needs to be entered in all resource transactions. Thus, as we have seven possible transactions in the pattern language, we expected to have seven hot spots concerning this feature. However, we found only four. The missing ones were for patterns 5, 9 and 10. Making a deeper domain analysis, we concluded that this feature is not desired for maintenance systems (patterns 9 and 10), because it is very rare to have resources to be maintained in quantities (they are usually unique). So only one new hot spot was added (number 12), as it makes sense to reserve more than one copy of a resource. It has been forgotten during the pattern language writing and, afterwards, the pattern language was fixed to include this requirement.

Hot spot 36 of the appendix is another example of a hot spot found during step *d2*, by analyzing class attributes that need to be computed. There are several different ways of computing the fine that customers need to pay when a transaction is paid after its due date. Examples are fixed daily, weekly or monthly fees, or a percentage of the total due value. Besides the fine, interests may optionally be charged. Hot spots 8 and 35 were also found during step *d2* of the section 3.7 process. It was not possible to identify hot spots during step *e* using the GRN pattern language because it refers to analysis patterns and is not concerned with design and implementation aspects. We have decided not to include hot spots of this category in a first version of the framework, but will consider this possibility in future versions (see discussion at the end of section 3.7).

Pattern 6: TRADE THE RESOURCE

6.1 Context

Your application deals with trade of resources, which may involve resources sold and/or purchased. You have already identified and Quantified these resources. Resource trading may be thought of as a resource property transference, in which a resource owned by one party becomes owned by another party. In a sale, if the resource is not available in stock, then the customer can fill in an order that will be granted when possible. In a purchase an order is made to the supplier who delivers the resource within a certain period.

6.2 Problem

How do you manage the resource trades made by your application?

6.3 Forces

- It is essential to log trade information, because it can be used to generate important reports on resource demand and organization gains (most systems in this domain are concerned with profits).
- The additional storage space and processing time required to log trade information has to be balanced against possible gains in system functionality when evaluating costs versus benefits. For example, it may be enough to increase and decrease stock levels when resources are traded, without considering other trade details.

6.4 Structure

Figure 17 shows the TRADE THE RESOURCE pattern.

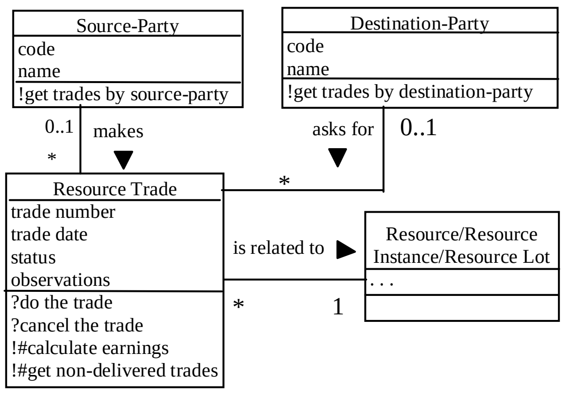


Figure 17: TRADE THE RESOURCE pattern

6.5 Participants

Resource Trade: represents all the details involved in trading the resource. The attribute `status` denotes the trade stage: pending, partially fulfilled, or fully fulfilled. When the MEASURABLE RESOURCE sub-pattern has been applied earlier, then an attribute `Quantity` is added to denote a non-unitary resource trade.

Resource/Resource Instance/Resource Lot: the choice among Resource, Resource Instance or Resource Lot depends on the quantification sub-pattern used.

Source-Party: represents the original resource owner, for example, in the case of a sale it is the organization department or branch that sells the resource, and in the case of a purchase it is the supplier organization. This class is optional for small sale systems where there are no departments or branches.

Destination-Party: represents the final resource owner, for example, in the case of a sale it is the customer buying the resource, and in the case of a purchase it is the organization department or branch buying the resource. This class is optional for small purchase systems where there are no departments or branches and also in systems where the customer is not logged, as in supermarkets.

6.6 Example

Figure 18 shows an instantiation of the TRADE THE RESOURCE pattern for an Inventory Control system.

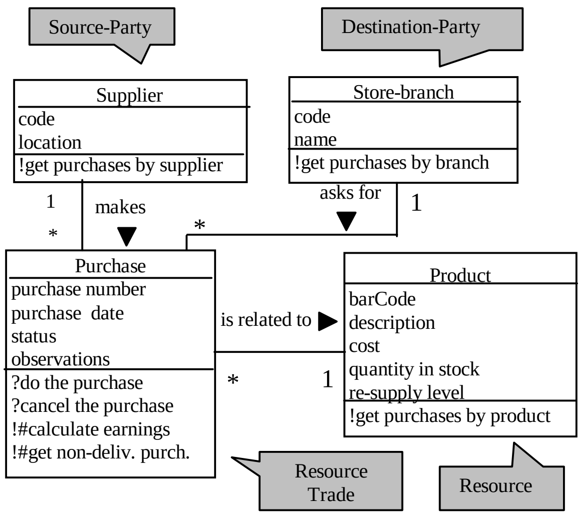


Figure 18: Instantiation of the TRADE THE RESOURCE pattern

6.7 Following patterns

Now, look at patterns in Section 2.3, which are useful for modeling other trade details. As a trade is followed by a delivery and can be preceded by a quotation, try to use the patterns QUOTE THE TRADE (7) and CHECK RESOURCE DELIVERY (8).

Figure 2 – Example of one pattern of the GRN pattern language

5. Concluding remarks and future work

It is rather intuitive that splitting the application domain in several patterns implies in the isolation of several hot spots. This makes the system composed of smaller parts, which need to be joined to make up the specific application. This fact was confirmed during the case study, where we have observed many hot spots identified from the pattern language graph, which means that optional parts of the system are isolated in patterns.

We have to remember that the pattern language construction involves an analysis of the application domain and having practical experience in the development of applications for this domain. In our case the first author had more than ten years of practice that allowed the development of the pattern language. During the framework usage in the development of specific applications and because of the application domain evolution, new hot spots may be necessary. In this case, the pattern language must also be updated, including new patterns or changing existing ones.

It could be argued that constructing a pattern language only to help in the identification of the hot spots of a possible framework would not be worth the effort. We think that this is not the case, because a pattern language can also be used for documenting the framework, as already shown in several works [1, 19]; for supporting the framework design and implementation [9, 10]; and as a method to guide the transformation of the framework in a concrete application [9].

We are interested in the definition of two different processes: the first to build frameworks based in pattern languages and the second for using a pattern language during the instantiation of applications with a framework built based in that pattern language. This second process may include the development of a wizard with an interface that follows the same concepts of the pattern language, so that users can apply the pattern language and have their systems semi-automatically built.

We also intend to propose a special notation to represent each hot spot identified from the pattern language. Our goal is to provide enough information about the hot spot to ease its subsequent design and implementation. For that, we are analyzing several proposals for hot spot representation found in the literature, among which are Pree’s hot spot cards [24], Schmid’s hot spots high level design specification [26, 27], Froehlich’s hooks [18] and Fontoura’s UML-F [16]. Besides that, we are studying whether it is important to represent the framework flexibility in the patterns of the pattern language, for example using UML-F extensions to UML [16] in the class diagrams of each pattern. At first glance it seems that this is desirable if the pattern language is being developed with the intent of building a framework based on it. Otherwise this would not be much useful.

References

[1] AARSTEN, A.; BRUGALI, D.; MENGA, G. (2000). A CIM Framework and Pattern Language, *in* “FAYAD, M. E. & JOHNSON, R. E. (eds.) (2000). *Domain-Specific Application Frameworks: Frameworks Experience by Industry*, John Wiley & Sons.”, p. 21-42.

[2] BOYD, L. (1998). Business Patterns of Association Objects. In: “R. Martin, D. Riehle, F. Buschmann (eds.) *Pattern Languages of Program Design 3*, Addison-Wesley, 1998”, p. 395-408.

[3] BOSCH, J. et al. (1999). Framework Problems and Experiences, *in:* “FAYAD, M. E.; JOHNSON, R. E.; SCHMIDT, D. C. (eds.) (1999). *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, John Wiley & Sons”, p. 55-82.

[4] BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO, P. C. (1998). A Family of Patterns for Business Resource Management. Proceeding of the 5th Annual Conference on Pattern Languages of Programs (PLOP’98), Monticello-IL-EUA. *Technical Report #WUCS-98-25*, Washington University in St. Louis, Missouri, USA, august, 1998,

available at 15/03/99 in the URL: jerry.cs.uiuc.edu/plop/plopd4-submissions/plopd4-submissions.html.

[5] BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO, P. C. (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.

[6] BRAGA, R. T. V. (2001). GREN: A framework for Business Resource Management. *WEB Page*, ICMC-USP, São Carlos-SP, Available in 04/10/01, URL: www.icmc.sc.usp.br/~rtvb/ GRENFramework.html.

[7] BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO, P. C. (2001). Extension of the Pattern Language for Business Resource Management. *Unpublished*, ICMC-USP, São Carlos-SP. Available for download in 03/10/01, URL: www.icmc.sc.usp.br/~rtvb/extended_patlang.zip.

[8] BROWN, K. & WHITENACK, B. G. (1996). Crossing Chasms: A Pattern language for Object-RDBMS Integration, The Static Patterns, *in* "VLISSIDES, J.; COPLIEN, J.; KERTH, N. (eds.) (1996). *Pattern Languages of Program Design 2*. Reading-MA; Addison-Wesley", p. 227-238.

[9] BRUGALI, D. & MENGA, G. (1999). Frameworks and Pattern Languages: an Intriguing Relationship. *ACM Computing Surveys*, march 1999.

[10] BRUGALI, D.; MENGA, G.; AARSTEN, A. (2000). A Case Study for Flexible Manufacturing Systems, *in* "FAYAD, M. E. & JOHNSON, R. E. (eds.) (2000). *Domain-Specific Application Frameworks: Frameworks Experience by Industry*, John Wiley & Sons.", p. 85-99.

[11] CAREY, J.; CARLSON, B.; GRASER, T. (2000). *SanFrancisco Design Patterns: Blueprints for Business Software*, Addison-Wesley.

[12] COAD, P.; NORTH, D.; MAYFIELD, M. (1997) *Object Models: Strategies, Patterns and Applications*, 2.ed., Yourdon Press.

[13] COPLIEN, J. O. (1995) A Generative Development-Process Pattern Language, *in* "Coplien, J.O. & Schmidt, D. C. (1995) *Pattern Languages of Program Design*, Addison-Wesley", p. 183-237.

[14] 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.

[15] GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. (1995). *Design Patterns – Elements of Reusable Object-Oriented Software*. Reading-MA, Addison-Wesley.

[16] FONTOURA, M. F.; PREE, W.; RUMPE B. (2000) UML-F: A Modeling Language for Object-Oriented Frameworks, 14th European Conference on Object Oriented Programming (ECOOP 2000), *Lecture Notes in Computer Science* 1850, Springer, 63-82, Cannes, France.

[17] FOWLER, M. (1997). *Analysis Patterns*. Addison-Wesley.

[18] FROEHLICH, G.; HOOVER, H. J.; LIU, L.; SORENSON, P. (1997). Hooking into Object-Oriented Application Frameworks. Proceedings of the International Conference on Software Engineering, Boston-Mass-USA, p. 491-501.

[19] JOHNSON, R. E. (1992). Documenting Frameworks using Patterns, Proceedings of the Conference on Object-Oriented Programs, Systems, Languages and Applications (OOPSLA'92), Vancouver, British Columbia, p. 63-76.

[20] JOHNSON, R. E. (1996). Transactions and Accounts, *in* "VLISSIDES, J.; COPLIEN, J.; KERTH, N. (eds.) (1996). *Pattern Languages of Program Design 2*. Addison-Wesley", p. 239-249.

[21] JOHNSON, R. E.; WOOLF, B. (1998). Type Object. In "Martin, R., Riehle D., Buschmann F. (eds.) Pattern Languages of Program Design 3, Addison-Wesley", p. 47-65.

[22] MESZAROS, G. (1996). A Pattern Language for Improving the Capacity of Reactive Systems, *in* "VLISSIDES, J.; COPLIEN, J.; KERTH, N. (eds.) (1996). *Pattern Languages of Program Design 2*. Reading-MA; Addison-Wesley", p. 575-591.

[23] PREE, W. (1995). *Design Patterns for Object-Oriented Software Development*, Reading-MA, Addison-Wesley.

[24] PREE, W. (1999). *Hot-spot-Driven* Development, in: "FAYAD, M. E.; JOHNSON, R. E.; SCHMIDT, D. C. (eds.) (1999). *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, John Wiley & Sons", p. 379-393.

[25] ROBERTS, D. & JOHNSON, R. E. (1998). Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks, *in Martin et al. (1998)*, p. 471-486, disponível em 12/03/99 na URL: st-www.cs.uiuc.edu/users/droberts/evolve.html

[26] SCHMID, H. A. (1997). Systematic Framework Design by Generalization. *Communications of the ACM*, v. 40, n.10, p. 48-51.

[27] SCHMID, H. A. (1999). Framework Design by Systematic Generalization, in: "FAYAD, M. E.; JOHNSON, R. E.; SCHMIDT, D. C. (eds.) (1999). *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, John Wiley & Sons", p. 353-378.

Appendix - GREN Framework hot spots

Hot spot number	Name	Description	Type	Source in the pattern language	Pattern #
1	Resource Qualification	A resource can have a type, but this is optional. It can also have multiple types or nested types.	PARTIC_ CHOICE	Participants, Variants	1
2	Resource Quantification	A resource can be unique, can have multiple instances, can be managed in quantities or in lots	PARTIC_ CHOICE	Participants, Structure, Variants (sub-patterns)	2
3	Resource Storage	It can or cannot be desirable that the application manages the resource storage	PATTERN _OPTION	Language Graph+ Context	3
4	Resource Rental	The application may or may not concern resource rental	PATTERN _OPTION	Language Graph + Context	4
5	Existence of Source-party* in the rental	The organization may be small and not have branches or departments	PARTIC_ OPTION	Participants	4
6	Rented instance entry	It is necessary to read the instance number of the rented resource when the rental refers to an instantiable resource (propagation of pattern 2 usage)	PROPAGA TION	Participants	4
7	Rented quantity entry	It is necessary to read the quantity of rented resources when the rental refers to a measurable resource (propagation of pattern 2 usage)	PROPAGA TION	Participants	4

* In business resource management systems a resource moves from a source-party (the party that initially owns the resource) to the destination-party (the party that acquires the resource, temporary or definitely).

Hot spot number	Name	Description	Type	Source in the pattern language	Pattern #
8	Instance number generation	The instance number of the resource to be rented can be supplied by the user or automatically generated by the system	BEHAVIO UR	-	4
9	Resource reservation	The application may or may not need to deal with resource reservation	PATTERN _OPTION	Language Graph + Context	4
10	Existence of Source-party in the reservation	The organization may be small and not have branches or departments	PARTIC_ OPTION	Participants	5
11	Reservation of the resource instance instead of resource	It may be allowed for the user to reserve a particular instance instead of reserving the resource	RELATIO NSHIP	Participants	5
12	Reserved quantity entry	It is necessary to read the quantity of reserved resources when the reservation refers to a measurable resource (propagation of pattern 2 usage)	PROPAGA TION	-	5
13	Resource trade	The application may or may not concern the trade of resources	PATTERN _OPTION	Language Graph + Context	6
14	Existence of Source-party in the trade	In sale systems, the organization may be small and not have branches or departments.	PARTIC_ OPTION	Participants	6
15	Existence of Destination-party in the trade	In purchase systems, the organization may be small and not have branches or departments.	PARTIC_ OPTION	Participants	6
16	Traded quantity entry	It is necessary to read the quantity of traded resources when the trade refers to a measurable resource (propagation of pattern 2 usage)	PROPAGA TION	Participants	6
17	Resource quotation	The application may or may not concern the quotation of resources	PATTERN _OPTION	Language Graph + Context	7
18	Existence of Source-party in the quotation	In sale systems, the organization may be small and not have branches or departments.	PARTIC_ OPTION	Participants	7
19	Existence of Destination-party in the quotation	In purchase systems, the organization may be small and not have branches or departments.	PARTIC_ OPTION	Participants	7
20	Quoted quantity entry	It is necessary to read the quantity of quoted resources when the quotation refers to a measurable resource (propagation of pattern 2)	PROPAGA TION	Participants	7
21	Check of resource delivery	The application may or may not need to take care of checking the resource delivery	PATTERN _OPTION	Language Graph + Context	8
22	Existence of Source-party in resource delivery	The organization may be small and not have branches or departments.	PARTIC_ OPTION	Participants	8
23	Delivered quantity entry	It is necessary to read the quantity of delivered resources when the delivery refers to measurable resources (propagation of pattern 2)	PROPAGA TION	Participants	8

Hot spot number	Name	Description	Type	Source in the pattern language	Pattern #
24	Resource maintenance	The application may or may not concern the resource maintenance	PATTERN _OPTION	Language Graph + Context	9
25	Tasks specification	The application may or may not want to individually specify the tasks involved in the maintenance	PATTERN _OPTION	Language Graph + Context	14
26	Parts discrimination	The application may or may not want to discriminate the parts used in the maintenance	PATTERN _OPTION	Language Graph + Context	14
27	Different executors for each task	The application may allow a different executor to perform each maintenance task	RELATIO NSHIP	Participants	15
28	Management of several resources in a single transaction	It may be desirable several resources to be managed in a single transaction	PATTERN _OPTION	Language Graph + Context	11
29	Management of the transaction payment	The application may or may not deal with the several payments associated to a transaction	PATTERN _OPTION	Language Graph + Context	12
30	Identification of the transaction executor	The application may treat commissions to be paid for the transaction executor	PATTERN _OPTION	Language Graph + Context	13
31	Commission according to installments paid	It may be desirable that commissions be paid to executors only when an installment is paid by the customer	BEHAVIO UR	Variants	13
32	Executor team	The application may allow transactions to be performed by executor teams so that commissions are split among them.	PARTIC_ CHOICE	Participants	13
33	Quotation Tasks specification	The application may or may not want to individually specify the tasks involved in the quotation	PATTERN _OPTION	Language Graph + Context	14
34	Quotation Parts discrimination	The application may or may not want to discriminate the parts used in the quotation	PATTERN _OPTION	Language Graph + Context	15
35	Charge of rentals	The application may have no charge for rentals, as in some libraries	BEHAVIO UR	-	4
36	Fine computing	The computing of fines due to delayed payment may vary from application to application	BEHAVIO UR	-	12