The Umbrella Pattern Language
Towards a Pattern Language for Analysis Patterns Integration

Haitham S. Hamza\textsuperscript{1} and Mohamed E. Fayad\textsuperscript{2}

\textsuperscript{1}Computer Science and Engineering Dept.
University of Nebraska-Lincoln
Lincoln, NE 68588, USA
hamza@cse.unl.edu

\textsuperscript{2}Computer Engineering Dept., College of Engineering,
San José State University
One Washington Square, San José, CA 95192-0180
m.fayad@sjsu.edu

ABSTRACT

Different analysis patterns may have different structures. Such structure inconsistency limits the reuse space of patterns. Consequently, the strength of analysis patterns will not be fully realized. In [7], a theoretical framework for the general integration problem of analysis patterns has been proposed. Our long-term goal is to develop a pattern language for integrating analysis patterns of different structures. In this paper, we focus on the particular problem of integrating stable analysis patterns with traditional patterns. We develop the Umbrella pattern language to address this particular problem.

1. Introduction

The increasing interest in analysis patterns in the last seven years has generated a wealth of analysis patterns that spans different domains such as health care, business, system security, and many others [2][3][4][13][14][15][16][17]. However, these analysis patterns are not all compatible, that is, different analysis patterns use different structures to present the pattern. For example, in [15], analysis patterns may have two layers, namely, the knowledge layer and operational layer, whereas stable analysis patterns (See Section 2) [6] follow the stability concepts [11], and hence, they have different layered structure. Other analysis patterns do not use layered structures such as those presented in [5] [17].

This structure inconsistency may affect pattern reusability adversely. Structure inconsistency complicates the integration of analysis patterns of different structures, and hence, the reuse space of the pattern may be diminished.

The problem of analysis pattern integration has been formally defined in [7], and a generic framework for pattern integration has been proposed and demonstrated.
Moreover, the formal definition of the problem of integrating stable analysis patterns with traditional patterns has been also introduced [7]. We extend this work by providing a pattern language for solving the integration problem.

Stable analysis patterns use the concepts of software stability [11]: Enduring Business Themes [1], Business Objects, and Industrial Objects. In this paper, we refer to any pattern that does follow the same structure as stable analysis patterns as a traditional pattern.

2. Paper History: The Big Picture

In PLoP 2002, we presented our early effort in developing a pattern language for stable analysis patterns [9]. This pattern language has been enhanced and extended in PLoP 2003 [18]. Figure 1 gives the big picture of the pattern language that we aim at for developing stable patterns. The objective of the overall pattern language is to cover the essential aspects related to concept of stable patterns.

We use a development-oriented approach to classify the patterns in our pattern language. In this approach, we define the main development phases for developing stable analysis patterns (shown in the black boxes in Figure 1). The process of developing stable patterns involves four main steps: Developing Stable Patterns, Documenting Stable Patterns, Testing and Validating Stable Patterns, and finally, Applying Stable Patterns. Each of these four steps is carried out through a collection of patterns. Our long-term goal then is to identify and document these patterns.

The Umbrella pattern language that we present in this paper fits into the fourth phase, Applying Stable Patterns. However, the Umbrella pattern language can be also used in the first phase, Developing Stable Patterns. As we shall see, the integration of stable analysis patterns with traditional pattern will result in a new stable analysis pattern, or more generally, an analysis model.
Towards a Pattern Language for Analysis Patterns Integration

Figure 1. Description of the overall pattern Language (Does not show all the patterns)
3. Notations

In this Section, we define the notation that we will be using in the rest of this paper. We follow the same notation identified in framework given in [7] to refer to stable patterns, traditional patterns, and their objects as following:

\( S \): refers to a stable analysis pattern,
\( T \): refers to a traditional pattern,
\( T^S \): refers to the stable version of the traditional pattern (explained later in the paper),
\( ST \): refers to the pattern resultant from integrating the two patterns \( S \) and \( T \).

\( S_{EBT} \): A set that contains the EBTs objects in the stable pattern \( S \),
\( S_{Bo} \): A set that contains the BOs objects in the stable pattern \( S \), and
\( S_{Io} \): A set that contains the IOs objects in the stable pattern \( S \).

The last three notations above apply also for the traditional pattern \( T \).

4. The Umbrella Pattern Language Meta-Pattern

In this Section, we document the Umbrella pattern language as a meta-pattern. The problem Section of this meta-pattern illustrates the overall problem of integrating stable analysis patterns with traditional patterns. Likewise, the forces of this meta-pattern represent the global forces of the integration problem. The solution part will be the overall Umbrella pattern language itself.

Pattern Name: The Umbrella Pattern Language

Problem
Stable analysis patterns have different structure than other existing patterns. Suppose that we want to analyze a new problem, and suppose that the problem can be analyzed by using stable analysis patterns and other patterns that do not follow the stability structure. The question is: how can we integrate stable analysis patterns and the traditional patterns to analyze the new problem?

Forces
The following are the global forces that are encountered by the Umbrella pattern language [7]:

1. Absence of EBTs: Traditional analysis patterns do not contain EBTs. This fact complicates the integration process. For one reason, in some situations, new EBTs should be identified to represent the core knowledge of the traditional patterns that are used in the integration. However, increasing the number of EBTs is not desired as
many EBTs would imply many concepts in the pattern, this makes the pattern a subsystem with less reuse opportunity. Another reason that makes the addition of EBTs undesirable is that experience shows that identifying the right EBT is not easy. A developer may run into a situation where wrong and/or unnecessary EBTs are identified.

2. **Hidden BOs:** Some traditional patterns consist of pure IOs. In such case, new BOs need to be identified. Moreover, the context of the problem we analyze may require the addition of new BOs as well. Another source that may lead to hidden BOs is the EBT-IO relationship that may exist when trying to integrate the two patterns. When new objects are introduced during the integration process, the relationships between these objects and the original objects of the integrated patterns should be identified. As discussed in Section 3, the structure of stable analysis patterns may introduce new BOs if an IO needs to be connected to an EBT in the integrated pattern. All these added BOs are called hidden BOs. Identifying BOs may not be straightforward.

3. **Overlapping of Objects in Different Layers:** In some situations, an object in the traditional pattern may play the role of an EBT in the stable pattern; however, this object does not fully satisfy the EBT characteristics, and hence, this object is not stable. In such case, the object may need to be renamed and modified to become an EBT. This "stabilizing" process may introduce some complication to the integration, as finding these objects and replacing them with an EBT is not straightforward.

**Solution**

To integrate stable analysis patterns with traditional patterns, we propose the umbrella pattern language. The key observation that we use to develop this pattern language is that any traditional pattern can be theoretically viewed as a stable pattern with zero EBTs, zero or more BOs, and at least one IO. A traditional pattern will probably consist of few BOs, and many IOs. However, there are two main practical extreme cases. One case is to encounter a traditional pattern that consists of pure IOs without any BOs, which indicates a highly unstable pattern. Another extreme case is a traditional pattern with pure BOs.

Patterns that constitute the Umbrella pattern language along with their relationships are depicted in Figure 2. A shown in the figure, we presume that we have analyzed the problem and we were able to identify two patterns that can be reused to analyze the problem, our problem now is how to interweave these patterns together to build the new pattern (or in more general sense, analysis model) that we need. Without lose of generality, we consider the simple case of integrating one stable pattern with one traditional pattern. The general case of integrating multiple stable patterns with multiple traditional patterns should follow the same approach.

The figure shows the two patterns $S$ and $T$ in dashed ovens, which form the two inputs of the Umbrella pattern language. The bold ovens highlight the four core patterns in the Umbrella pattern language: Create stable version of $T$, Identify BOs of ST, Identify EBTs of ST, and Identify IOs of ST. These core patterns will be used whenever we integrate two patterns $S$ and $T$, and hence they are considered the core patterns. Other patterns that are
presented in ovens other than dashed or bold ovens represent patterns that may or may not be used during the integration process, depending on the integrated patterns structures. Three kinds of transition between patterns are used: the **dashed arrows** to indicate a transition between an input and a main pattern, the **unidirectional solid arrows** to indicate a transition between any two patterns in the specified direction only, and finally, the **double directional solid arrows** to indicate the possibility of transition between two patterns in two directions. The four main patterns in the Umbrella pattern language (the ones in the bold ovens), can be viewed as the four main phases of integrating stable patterns with traditional pattern. The first phase is a primarily phase and it is used to transfer the pattern $T$ into a virtual stable pattern. Phase 2, 3, and 4 produces the BOs, EBTs, IOs of the integrated pattern $ST$, respectively.

![Diagram](image-url)

*Figure 2. The Umbrella pattern language overview*
5. The Umbrella Pattern Language Description [In Progress]

In the following we describe in more details some of the patterns in the Umbrella pattern language shown in Figure 2.

**Pattern 1: Create Stable Version of the T pattern:**

**Problem:**

Given a traditional pattern $T$, how can we classify the objects of this pattern into BOs and IOs, so that we can create a virtual stable pattern $T^S$?

**Forces:**

1. Traditional patterns are built without the intention of differentiating between IOs and BOs. Therefore, it is possible that we come across an object that has some characteristics of both IOs and BOs. In this case it is hard to decide to which category this object belongs.

2. Sometimes, the chosen names for objects in the traditional patterns mislead the classification of these objects. We may come across a situation where the naming and convention of the object indicates an IO, while the semantic of the object is more an BO.

3. There are some situations where an object in the traditional pattern posses many of the EBTs characterize; yet, the object is not an EBT. Unfortunately, these specific kinds of objects are usually confusing as it is not clear whether it should be classified as an IO or a BO.

**Solution:**

The first rule of thump is that every traditional pattern should be viewed as a collection of BOs and IOs, without any EBTs, and hence, we can eliminate the possibility of any confusion that can result from objects that “look like” EBTs but they are not really EBTs. After that, each object in the $T$ pattern should be examined carefully against the main characteristics of BOs and IOs given in [12]. An object that has overlapping characteristics and can be classified as both an IO and BO should be decomposed into two objects; a BO and an IO.

If there is still confusion about classifying the object, a scenario-based approach can be then used, where the traditional pattern will be put in context and the characteristics of the object in this pattern will be observed more closely. The output of this phase is two sets, $T_{BO}$ and $T_{IO}$, the set of the BOs and the set of IOs in $T$, respectively. These two sets are used as inputs to other patterns as we shall see below.
Pattern 2: Identify BOs in $ST$:

Problem:
Given the set of BOs in the $T$ pattern $T_{Bo}$ (obtained from the pattern above), and the set of BOs in the $S$ pattern $S_{Bo}$, we want to identify the list of business objects in the integrated pattern $ST_{Bo}$.

Forces:
This step forms the core of the pattern integration process. There are many forces that complicate this step:
1. **Object Redundancy**: When integrating two analysis patterns, it is anticipated that they may share several objects. Such overlap between objects needs to be eliminated to avoid redundancy. Even though this objective sounds straightforward, it might be further complicated if such overlap is hidden within the semantic of the objects.
2. It is almost impossible to correctly identify all BOs of the resultant pattern at once. This is because of the fact that new EBTs or IOs might be added to the integrated to the pattern and this may require the addition of the new OBs.

Solution:
The objectives of this phase are: to remove duplicate BOs in the two sets, $S_{Bo}$ and $T_{Bo}$ if any, to unify similar BOs into one BO, to split a BO into two or more BO, and/or introduce new BOs if needed. The need of all, some, or none of these objectives depends on the nature of the $T$ and $S$ patterns as well as the nature of the desired integrated pattern and the context of the problem we want to analyze. This phase is conducted by examining the two sets of BOs, and then identifying the similarities and differences between these two sets and whether these similarities are in the semantic or the syntax of the objects. The approach that can be used to identify the BOs of the integrated pattern $ST$ depends on the kind of similarities between BOs in both $S$ and $T$. We can classify these kinds of similarities as following:

(a) **Similar Syntax and Similar Semantic**: This situation is straightforward and easy to handle. It occurs when two or more objects have the same name and same role. A common example of such case is happen with the common object *Party*, which may represent a human, an organization, etc.

(b) **Different Syntax and Similar Semantic**: two or more objects may have the same purpose semantically, while each has a different name. For example, a pattern may refer to a product with the object *Item*, while another pattern may use the object *Entity* instead.

(c) **Similar Syntax and Different Semantic**: This is situation is the opposite of the situation (b). Here, two or more object may have the same name but are semantically different.

(d) **Different Syntax and overlapping Semantic**: A more complex situation arises when two or more objects have different name but overlap partially in their semantic.
We intend to present a pattern for each of the above four cases. These patterns are the *Unify BOs*, *Eliminate BOs*, *Decompose BOs*, and *Add New BOs* patterns.

**Pattern 3: Identify EBTs in ST:**

**Problem:** Given the set of BOs in ST pattern $ST_{BO}$ (obtained from phase-2 above), and the set of EBTs in $S$ pattern $S_{EBT}$. The objective of this phase is to link the $ST_{BO}$ to the $ST_{EBT}$ set, and to identify the new EBTs, if any.

**Forces:**

- EBTs should capture the core aspects of the problem; however, some EBTs capture the core knowledge of the problem within a specific context. Such EBTs should be discarded from the model.
- Being expert in the domain does not always guarantee an accurate generation for the relevant EBTs. For instance, a professional chef might mistakenly identify pots, pans, and refrigerator as an enduring business theme for modeling the kitchen while they are in fact Industrial Objects (IOs) [7]. Thus, experience is essential but not sufficient condition for extracting the correct EBTs in the problem we analyze.
- Even though many of the selected EBTs might appear strongly related to the problem at first glance, many of them in fact have nothing to do with the problem being modeled.

**Solution:** This phase is conducted by examining the set of BOs in the system, and links them to the set of EBTs in the $S_{EBT}$. If the integration of $S$ and $T$ does not introduce a new concept to the system, then no new EBTs are needed. However, on the other hand, if new concepts are introduced by either $S$ or $T$, one or more EBTs may be needed. The pattern we proposed for identifying EBTs in [9] can be used to conduct this phase. If new EBTs are identified; the feedback to phase 2 is used to refine the BOs set obtained so far as it is possible that we need to identify new BOs in this case ("hidden BOs").

**Pattern 4: Identify IOs in ST:**

**Problem:**

Given the set of BOs in ST pattern $ST_{BO}$, the set of IOs in the $S$ pattern $S_{IO}$, and the set of IOs in the $T$ pattern $T_{IO}$. How can we identify the set of IOs for the ST pattern?

**Forces:**

1. Some traditional patterns contain several IOs that represent the unstable objects of enduring concepts that need rather to be presented as EBTs instead of IOs.
2. Some IOs need to be eliminated and abstracted as BOs instead.
3. IOs of traditional patterns may be connected to BOs that have been eliminated, unified, or decomposed into several BOs. It is not always easy to find the
relationship between IOs and BOs after such modifications. We need to ensure the same semantic of the original patterns in order to ensure correct models.

**Solution:** It is worth to note that the pattern $S$, in most cases, has no IOs. This is due to the fact that IOs are not stable objects whereas stable patterns aim at identifying the stable objects only (EBTs and BOs) in the problem. Therefore, in most cases the IOs in the system are those of the $T$ pattern. The result of applying this pattern might be one or more of the following actions a new IO(s) is added, and/or an IO(s) is removed, renamed, or integrated with another IO(s).

We need to match the set of BOs in $ST$, $ST_{BO}$ with the IOs sets in the $S$ and $T$ patterns, $S_{IO}$ and $T_{IO}$, and then identify the relationships between the two IOs sets as well as the IOs set and the $ST_{BO}$ set. When an IO is need to be connected to an EBT, in this case a new BO(s) is needed to establish this relationship.

6. Examples** [We want to add another example]

In this section we illustrate the use of the Umbrella pattern language through examples.

**Example I: Building a Resource Rental Pattern with a Negotiation Capabilities**

In this example, the objective is to integrate two patterns $S$ and $T$, where $S$ is the Negotiation stable analysis pattern [6], [13] shown in Figure 3, and the $T$ is the Resource Rental pattern [17] shown in Figure 4.

The integrated pattern, $ST$, is a pattern that can be reused to rent an entity while the negotiation process may be conducted to set an agreement on the entity renting aspects (e.g. price, time, etc.). The $ST$ pattern that result from the integration above has several practical applications. For instance, one may negotiate the standard price of renting a car if she will rent the car for a month or so. We obtain the integrated pattern $ST$ by using the framework proposed in Section 7.

First, the traditional pattern $T$ is classified into two sets of objects, $T_{Bo}$ and $T_{Io}$. Second, the $T_{Bo}$ along with the $S_{Bo}$ are used to identify the BOs of the integrated pattern $ST$, $ST_{Bo}$. In this phase, we unify the two BOs: AnyAgreement and ResourceBooking into the BO AnyAgreement. In some sense, AnyAgreement can be viewed as a more general form of booking, as if there is no negotiation process is performed, we assume that by booking the resource, the customer agrees on the conditions and terms that are posed by the renter. On the other hand, the renter agrees to rent the entity for the specified time and money. This booking process is nothing but an agreement between the renter and the customer. Therefore, the BO AnyAgreement can be internally adapted to accommodate both the booking and the agreements specifications defined in the $T$ and the $S$ patterns, respectively. In this paper, we do not show such adaptability; however, to show how the AnyAgreement can be adapted, we need to consider the internal structures of both AnyAgreement and ResourceBooking objects, and then merge both of them to form a

---

Copyright © 2004 Haitham Hamza, and Mohamed E. Fayad.
Permission is granted to copy for the PLoP 2004 conference. All other rights reserved.
modified AnyAgreement BO, which is one of the elements in the $ST_{Bo}$ set (That is to say, the BO AnyAgreement in $ST_{Bo}$ has different internal structure than that in the $S_{Bo}$.)

Since the negotiation and renting are two different concepts, integrating these two patterns requires the addition of a new EBT, Renting, as shown in Table 1. The Renting EBT forms an enduring concept in the new pattern $ST$. In this example, the iterative link between phase 2 and phase 3 will not result in any new BO; however, it is important to revisit phase 2 to define the relationship between BOs and the new EBT Renting.

Now we identify the IOs in the $ST$ pattern. This is easy to define in this example as there are IOs in the $T$ pattern only. In this phase it is important to identify the relationships between the different IOs and the BOs. As shown in Figure 5, the IO Customer, which was originally in the $T$ pattern, is now connected to the BO AnyParty, which belongs to the $S$ pattern. In this example, the iterative link between phase 4 and phase 2 results in a new BO Receipt. This new BO appears because the IO Payment should be included in the $ST$ pattern; however, the $S$ pattern does not include any object that is related to the payment, and hence, a new BO is added in this case to handle the IOs Payment in the model.

![Figure 3: Resource Rental pattern](image)

Copyright © 2004 Haitham Hamza, and Mohamed E. Fayad.
Permission is granted to copy for the PLoP 2004 conference. All other rights reserved.
**Figure 4:** Negotiation pattern stable object model

**Figure 5:** The resultant integrated pattern
7. Consequences of Applying the Umbrella Pattern Language

In this Section we highlight both positive and negative impact of the Umbrella pattern language.

Negative consequences:

(-) The use of the Umbrella pattern language to integrate patterns may lead to a traceability problem. Once the two patterns are integrated, it becomes hard, if not impossible, to identify the original patterns that have been used. One solution to this problem might be an exhaustive documentation of the integration process itself. However, we see that this solution may sound practical for small-scale projects; however, for large-scale projects, it may become less efficient to document and maintain every integration process within the system. Currently, we investigate this problem more closely and we try to propose a solution that avoids the drawbacks of the direct integration that we propose in this paper.

(-) There is no guarantee (no formal validation) that the resultant pattern does not include redundant objects that have the same semantic. Also, we can guarantee that the resultant pattern would work effectively. However, we argue that, this step would reduce the complexity of developing analysis models from scratch, and in the worst case, it forms a starting point that developers can use to start their analysis.

Positive consequences:

(+) The Umbrella pattern language allows for virtually integrates any analysis patterns with stable analysis patterns. Therefore, we can extent the space of stable analysis patterns reusability to include any available analysis pattern.

(+) The integration of analysis patterns can help reduce the redundancy that currently exists in analysis patterns (several analysis patterns address the same exact problem). This will save time and effort for performing analysis.

5. Conclusions and Future Work

In this paper we propose we show how stable analysis patterns can be integrated into traditional pattern to construct a new useful pattern, or more generally, an analysis model. The importance of this approach lay in the fact that by integrating patterns of different structures, virtually, we widen the selection space from which a developer can choose and reuse patterns.

One problem that we consider in our future work is to solve the traceability problem that our framework may introduce. Once the two patterns are integrated, it becomes hard, if not impossible, to identify the original patterns that were used in the process.

The proposed pattern language is a one-way approach. It does not show how to integrate stable analysis patterns into traditional patterns such that the resultant pattern can fit within the traditional structure of the system. The result of this approach is a stable
pattern which may not fit with the other patterns used in the system. We currently investigate this limitation and its possible solutions, as a starting point to realize such generic framework, we investigated the global forces of the general analysis pattern integration problem, in addition, we formally define the problem in order to understand and capture the aspects that are involved in such integration [7].

Acknowledgements

We would like to thank members of our workshop at PLoP 03 for their effort in reading and providing valuable comments: Eduardo Fernandez, Joe Yoder, Diethelm Bienhaus, Lei Zhen, and Michael Weiss. Also, we thank our PLoP 03 shepherd, Marshall Cline, for his discussions and comments during and after the shepherding process, and for his review of the pattern language draft for PLoP 2002.

We would like to thank Bruce Whitenack, the shepherd of our PLoP 2002 pattern language, for his useful comments and for his valuable suggestions.

Thanks to the members of the PLoP '02 Workshop, Sweet Home Allerton, for taking time to review the proposed patterns: Ahmed Mahdy, Linda Rising, Danny Dig, Joel Jones, Shasha Wu, Toby Sarver, Masao Tomo, Joe Yoder, and. Martine Devo.

References

[3] E.B. Fernandez, "Stock Manager: An analysis pattern for inventories". In 7th Pattern Languages of Programs Conference (PLoP’2k), Monticello, IL, USA