On the Integration of Stable Analysis Patterns with Traditional Patterns

Haitham S. Hamza

Computer Science& Engineering Dept., University of Nebraska-Lincoln,
Lincoln, NE 68588, USA
hhamza@ cse.unl.edu

Abstract. In analysis phase, conceptual models are developed to understand and communicate the core knowledge of the problem. Patterns have emerged as a promising technique for software reuse. Patterns that represent conceptual models are called analysis patterns. Analysis patterns can be reused to understand similar and related problems. Real-life problems may require the integration of several analysis patterns. However, a fundamental limitation in analysis patterns today is the lack of structure consistency, i.e. different analysis patterns have different structures. Such inconsistency may hinder the integration of analysis patterns, and hence, it limits the reuse space of patterns. In previous work [9], we have formally defined the general problem of integrating analysis patterns. In this paper, we focus on the particular problem of integrating stable analysis patterns with traditional patterns. We refer to this problem as the ***S-T Integration*** Problem. We define the problem and propose an approach to solve it. We demonstrate the proposed approach through two case studies.

1 Introduction

Analysis phase forms the base upon which the rest of the development cycle is built. The obscurity of doing accurate analysis along with the fact that analysis is a costly and a time-consuming activity; both make the development of effective and reusable analysis artifacts of a great interest in software engineering community.

The last few decades have witnessed a significant evolution of several software reuse approaches, including *Model-Based software*, *Component-Based Software Engineering (CBSE)*, and *software patterns.* Despite their differences, all approaches share the same goal of enabling the concept of reuse within the different phases of software development.

Recently, patterns have received an increasing interest as a promising technique for software reuse. Patterns can allow for improving the quality and reducing the cost and time of software development [1] [3]. In general, a pattern can be defined as: "An idea that has been useful in one practical context and will probably be useful in others" [17]. Further, analysis patterns can be defined as conceptual models that can be reused to analyze similar and related problems.

The increasing interest in analysis patterns over the last seven years has generated a wealth of analysis patterns that spans different a wide spectrum of domains such as health care, business, system security, and many others (e.g. [2][4][12][17][18][19]). However, these analysis patterns are not always compatible in the sense that different analysis patterns may have different structures. For example, in [17], an analysis pattern may have two layers, namely the knowledge layer and operational layer. Whereas stable analysis patterns (See Section 3) [7] [10] follow the structure of software stability models [14], which divide the pattern structure into three layers, namely, EBTs layer, BO layers, and IO layers (defined in Section 3). Other analysis patterns do not have layered structures such as those patterns presented in [6] and [19].

This structure inconsistency may adversely affect pattern reusability in different ways. First, structure inconsistency complicates the integration of analysis patterns of different structures, and increases the risk of misinterpretation of the pattern semantics. As a result, the expected benefits of reusing patterns may not be achieved. Second, new patterns will evolve that solve similar problems but have different structures, which increase patterns redundancy. Such patter redundancy may translate into more cost and time in the development cycle as more candidate patterns will appear, while in fact, these different patterns might be "semantically" similar. Third, structure inconsistency limits the space of patterns reuse, as reusability of specific pattern structure is more likely to be limited to those patterns that follow the same structure.

These problems may be avoided if we can develop an approach for integrating patterns of different structures. We refer to the problem of using analysis patterns of different structures to form a new analysis pattern, or more generally an analysis model, as the ***analysis patterns integration problem***. The problem of analysis pattern integration has been formally defined in [7]. In [7], a generic framework for pattern integration has been proposed and demonstrated. The problem of pattern integration (or composition) has been addressed in the context of design patterns. A summary of composition approaches for design patterns can be found in [20]. However, to the best of our knowledge, the problem of addressing the integration of analysis patterns of different structures has not been addressed before.

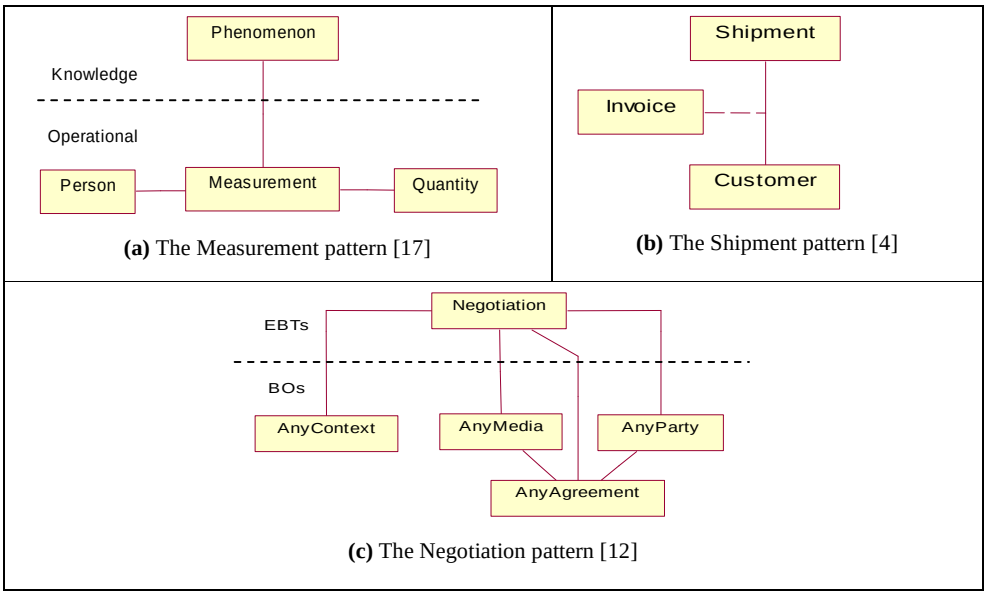
In this paper, we focus on a specific part of the general analysis patterns integration problem. This part is related to the integration of stable analysis patterns with traditional patterns. For the rest of this paper, a traditional pattern refers to any pattern that does not follow the stable analysis patterns structure defined in [7].

The remainder of this paper is organized as follows: An overview of the problem and our motivation is given in Section 2. An overview of stable analysis patterns is provided in Section 3. In Section 4 we discuss the challenges in integrating stable analysis patterns with traditional patterns S-T Integration Problem. The S-T Integration Problem notations and definition are given in Section 5. A high-level iterative approach to solve the integration problem is discussed in Section 6. In Section 7 we provide a case study for pattern integration. A discussion is given in Section 8; we conclude in Section 9.

2 Problem Overview and Motivation

Analyzing the problem from scratch is both tedious and costly task. In addition, novice developers may lack the necessarily experience for developing 'good' analysis models for their problems. The fact that analysis forms the base for any software development process, along with the fact that errors in analysis models may be discovered late in the development cycle where it is too late to avoid cost and time penalties, together make the idea of reusing analysis artifacts of a great interest in software community. Analysis patterns have emerged as a mean for reusing analysis models. Real-life problems are usually complex and more likely they will require the integration of several analysis patterns. However, these analysis patterns are not compatible in the sense that different patterns are developed using different approaches and methodologies. As a result, the structure of these patterns may vary greatly. This variation in pattern structures may complicate the reuse of these analysis patterns.

Figure 1 shows three analysis patterns with different structures. Figure 1 (a) shows the Measurement analysis pattern [17]. This pattern consists of two layers: the knowledge layer and the operational layer. In Figure 1 (b), the Shipment pattern is given [4]. As oppose to the previous two analysis patterns, this analysis pattern has no layers.
In Figure 1 (c), the Negotiation analysis pattern is shown [12]. This pattern consists of two layers, namely, the EBT layer and the BO layer.



(a) The Measurement pattern [17]

(b) The Shipment pattern [4]

(c) The Negotiation pattern [12]

Fig. 1. Analysis patterns with different structures: **(a)** A pattern with two layers (knowledge and operational) **(b)** the patterns has no layers **(c)** A pattern with two layers (EBTs and BOs).

Our long-term goal is to understand the different structures of existing analysis patterns and to develop a methodology that allows for the integration of analysis patterns of different structures. In this paper we focus on a specific part of the general integration problem of analysis patterns. This part is related to the integration of stable analysis patterns and traditional analysis patterns.

3 Stable Analysis Patterns: A Background

Software stability approach [14] is a layered approach for developing software systems. In this approach, the classes of the system are classified into three layers: the Enduring Business Themes (EBTs) layer [13], the Business Objects (BOs) layer, and the Industrial Objects (IOs) layer. Based on its nature, each class in the system is classified into one of these three layers.

EBTs are the classes that present the enduring and core knowledge of the underlying industry or business. Therefore, they are extremely stable and form the nucleus of the stable model. BOs are the classes that map the EBTs of the system into more concrete objects. BOs are semi-conceptual and externally stable, but they are internally adaptable. IOs are the classes that map the BOs of the system into physical objects. For instance, the BO "Agreement" can be mapped in real life as a physical "Contract", which is an IO. The detailed properties of EBTs, BOs, and IOs are discussed in [15] [16].

Figure 2 illustrates the concepts of stable models and their use. The EBTs and BOs in the left-hand side of the figure form a stable core. By ***externally*** adapting the BOs, through hooks, and by adding the necessary IOs, two new systems were constructed, system 1 and system 2, as shown in the right-hand side of the Figure. It is important to point out the assumption that these two systems are related (e.g. share the same domain) that we can identify common EBTs and BOs. The figure illustrates the important aspects of stability models: EBTs are highly stable and do not change from one system to anther, BOs are externally stable but are internally adaptable, and IOs are unstable and differ from one system to another.

Stable analysis pattern [7] [10] is a new approach for developing patterns by utilizing software stability concepts described above. The concept of stable analysis patterns was proposed as a solution for the limitations of contemporary analysis patterns discussed in [7]. The goal of stable analysis patterns was to develop models that capture the core knowledge of the problem and present it in terms of the EBTs and the BOs of that problem. Consequently, the resultant pattern will inherent the stability features and hence can be reused to analyze the same problem whenever it appears.

4 Challenges Faces the S-T Integration Problem

In this Section, we identify the forces that face the integration of stable analysis patterns with traditional patterns. The general challenges of integrating analysis patterns

in general are summarized in [9]. We summarize the challenges of integrating stable analysis patterns with traditional patterns as following:

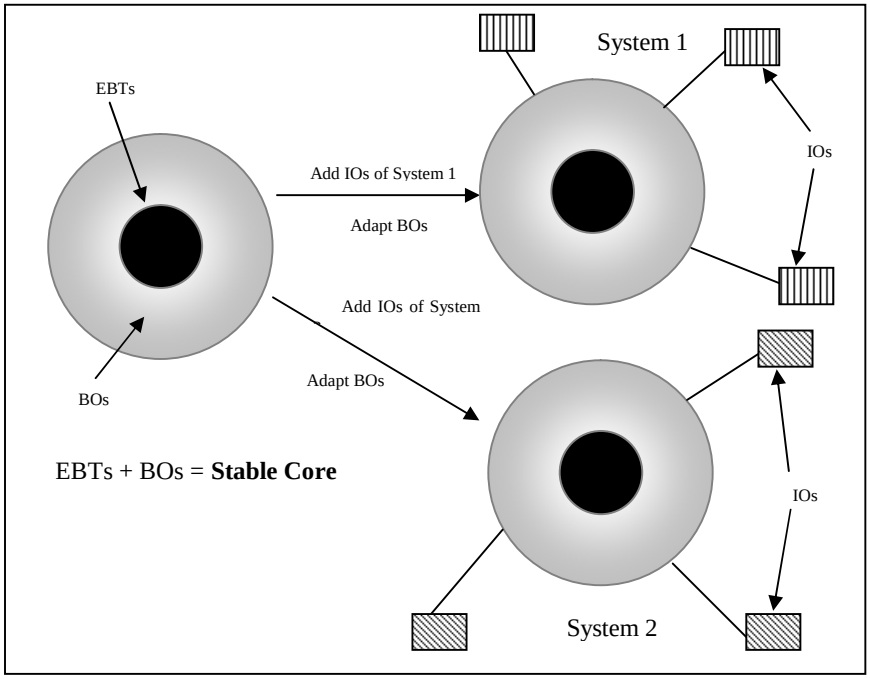


Fig. 2. Software stability Concept. The stable core (in the left) has been adapted (the BOs) and extended (the IOs) to construct to systems (in the right)

1. **The Absence of EBTs:** All traditional analysis patterns do not contain EBTs obejcts. 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 instead of a simple pattern, and hence reduces its reuse opportunity [10]. Another reason that makes the addition of EBTs undesirable is that our experience shows that identifying the right EBT is not easy. A developer may run into a situation where wrong and/or unnecessary EBTs are added to the resultant pattern.

2. **The Hidden BOs:** Some traditional patterns consist of IOs only. In this case**,** the absence of OBs may require the addition of new BOs. Moreover, the context of

the problem we analyze may require the addition of new BOs as well. All these added BOs are called hidden BOs. Identifying BOs may not be straightforward.

3. **The Overlapping in Different Layers:** In some situations, an object in the traditional pattern may play the role of an EBT; however, it 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 be used as an EBT. This "stabilizing" process may introduce some complication to the integration process, as finding these objects and replacing them with an EBT require close examination of each object and its role.

4. **The EBT-IO Relationship Identification:** 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.

5. **5. Detecting 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. This problem is a generic problem that faces the integration of any analysis pattern [9]. However, the structure of stable analysis patterns makes it less obvious to detect object overlapping. The detection of object redundancy might be complicated if such overlapping is hidden within the semantic of the objects. We identify four general types of object overlapping:

(a) Syntax and Semantic Overlapping: 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. This object appears in many patterns, and hence, when these patterns are to be integrated, these objects will become one object, perhaps with different roles or inheritance structure to capture the different kinds of party involved in the system.

(b) Different Syntax with Overlapping 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 as item, while other pattern may use the object Entity instead. Depending on the context of the system we analyze, the developer may choose the name of the object that she want as long as the two overlapping objects have the same meaning in the context of the problem. The complexity of this kind of object overlapping arises from the fact that misinterpretation of any of the two patterns may result in wrong action, for example, two objects may be unified because their role looks similar; however, this might not be true. Since handling this case depends on the user experience and knowledge, there is not guarantee that the situation will be handled correctly.

(c) Syntax Overlapping with 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. These objects should be both represented in the new pattern, but renaming should be done. The same complication discussed in the previous situation applies here.

(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. One approach to tackle this problem is to replace these objects with a new object that captures the semantic of the overlapped objects. However, this solution may not be appropriate when the objects have limited overlap. In such case, one may think of identifying a new object that captures the similarities and eliminate these overlapping from the other objects. The result will be a new object added to the original overlapping objects.

5 Notations and Problem Definition

In this Section, we define the notation that we will be using for the rest of this paper to discuss the ***S-T Integration Problem***. We follow the same notation identified 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 stable version of the traditional pattern (this will be explained),

***ST*:** the pattern that is resultant from the integration of 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 three last notations above apply for the traditional pattern *T* as well.

The ***S-T Integration*** Problem is defined as follows: Given a stable analysis pattern ***S*** and a traditional analysis pattern ***T,*** construct a new stable analysis pattern (or model) ***ST*** such that ***ST*** complies with stable analysis pattern structure and constraints.

6 A High-Level Iterative Approach for the S-T Problem

To integrate stable analysis patterns with traditional patterns, we propose a high-level iterative approach shown in Figure 3. The key observation that we use to develop our approach 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 situation is a traditional pattern with pure BOs.

In the following we describe the different phases of the integration approach:

Phase 1: *T-Classification Phase*: The input to this phase is the traditional pattern *T*. Each object in the *T* pattern is then examined carefully against the main characteristics of BOs and IOs given in [15]. Therefore, each object in *T* will be classified as either an BO or an IO. 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 phases as we shall see below.

Phase 2: BOs Identification phase: The inputs to this phase are: the set of BOs in the *T* pattern T_{Bo} (obtained from the first phase above), and the set of BOs in the *S* pattern S_{Bo}, which is given. 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. It is worth to point out that this phase can be iterative as indicated by the dashed line in Figure 3. In most cases, when this phase is first conduct in the integration process, BOs may be modified, eleminated, or unified. However, it is the iterative part that feedbacks from the third and fourth phase, that usually results in introducing new BO to the model (note that the unification of two BO may result in new BO; however, by new BO here we mean a totally new BO that has not been presented in any way in either *S* or *T*). 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. While we don't provide a systematic approach to test such similarities; however, we emphasize that both syntax and semantic similarities should be examined. Based on the result of this examination, the developer can decide what shall be done to similar objects, if any. For instance, syntax similarities might be simply resolved by renaming some of the objects; whereas, semantic similarities would require more involved approach to resolve the problem. The output of this phase the set, possibly the final set, of BOs in the integrated pattern *ST*, ST_{Bo}

Phase 3: EBTs Identification phase: The inputs to this phase are: 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. 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 [11] 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"). (note that you can give a flow chart with decision icons to describe the approach)

Phase 4: IOs Identification phase: The inputs to this phase are: 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}. In this phase, we identify the required IOs, if any. 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. In this phase several actions may occur: a new IO(s) may be added, and/or an IO(s) may be removed, renamed, or integrated with another IO(s). This phase can be conducted conducted by matching the set of BOs in *ST*, 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. The feedback between phase 4 and phase 2 is used in two cases: when new IOs are identified in phase 4, and when an IO is need to be connected to an EBT, in this case a new BO(s) is needed to establish this relationship.

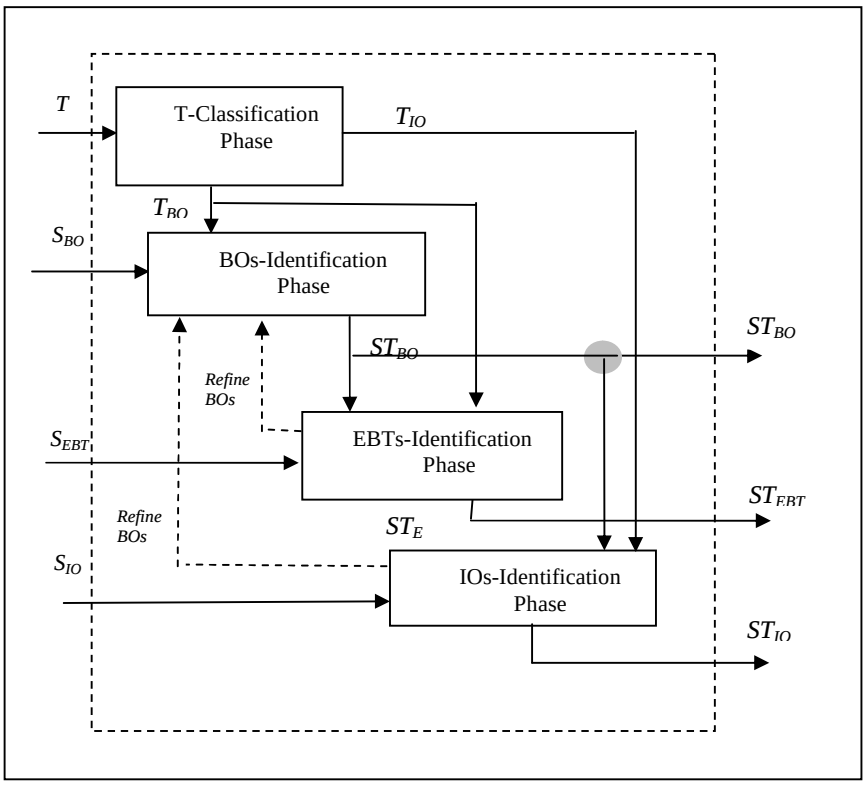


Fig. 3. The high-level integration approach to integrate the two patterns *S* and *T*.

7 Examples of Integrating Analysis patterns

In this section we illustrate the use of the proposed approach in Section 6 through two examples.

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

In this example the objective is to integrate two patterns *S* and *T*, where the *S* pattern is the Negotiation stable analysis pattern [12] shown in Figure 4, and the *T* pattern is the Resource Rental pattern [19] shown in Figure 5.

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 approach proposed in Section 5.

First, the traditional pattern *T* is classified into two sets of objects, T_{Bo} and T_{Io} .In the second phase, 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 ***externally*** 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 ***modified*** AnyAgrement 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}.)

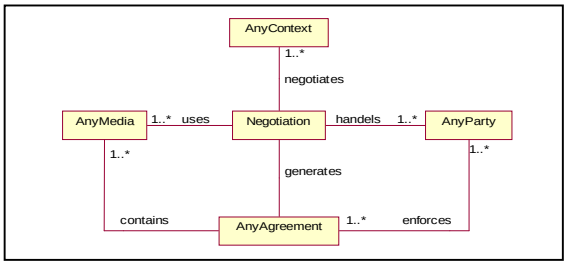


Fig. 4. *Negotiation* pattern stable object model [12].

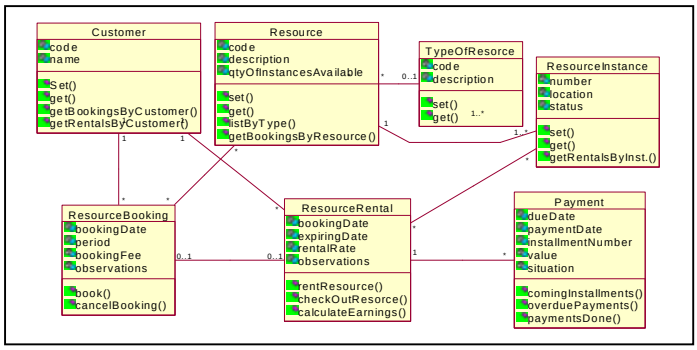


Fig. 5. Resource Rental pattern [19].

Since the negotiation and renting are two different concepts, integrating these two patterns requires the addition of a new EBT, *Renting*. 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 6, 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

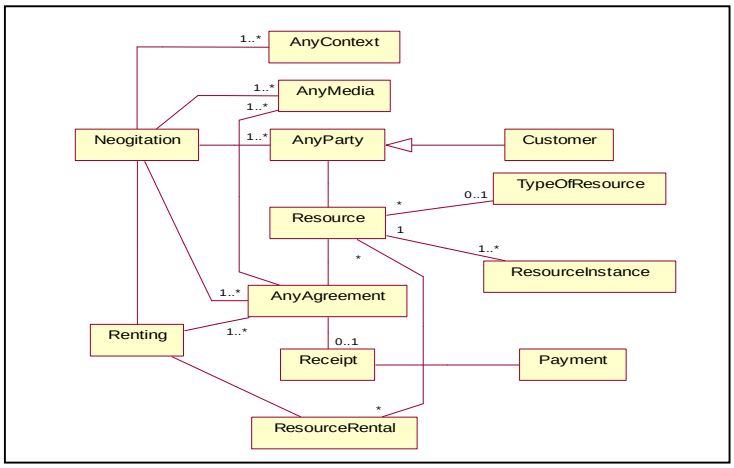


Fig. 6. The resultant integrated pattern

7.2 Example-2: *The Order and Shipment with Trust*

In this example we want to create a model for the order and shipment of a product with establishing a trust component. We use the Trust analysis patterns [21] (Figure 7), and the Order and Shipment patterns [4] (Figure 8). The order and shipment patterns are traditional patterns (*T*), while the Trust pattern is a stable analysis pattern (*S*). The details of integrating these two patterns follow the same approach of example-1, and hence are omitted due to space limitation. The integrated pattern is shown in Figure 9.

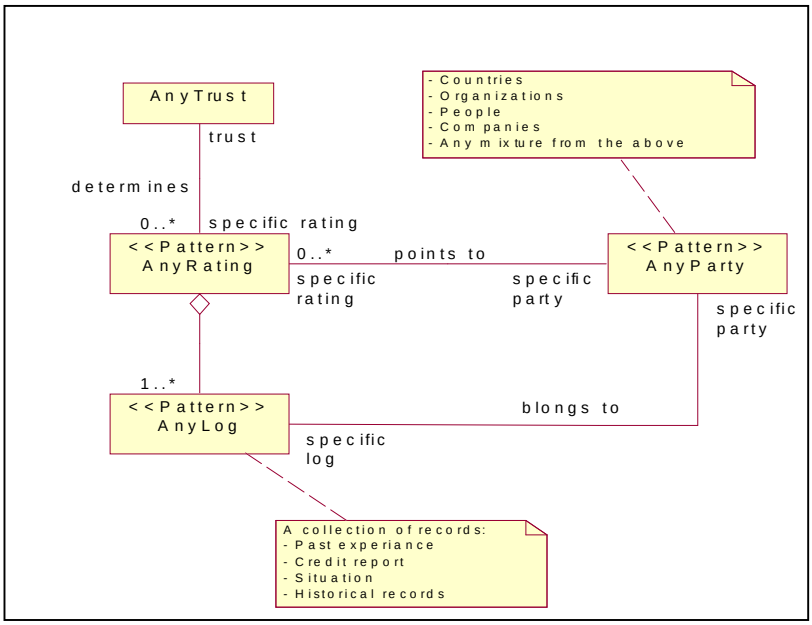
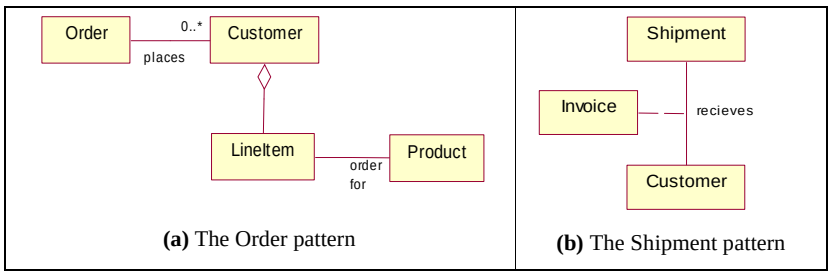


Fig. 7. The Trust analysis pattern [21]



(a) The Order pattern

(b) The Shipment pattern

Fig. 8. Analysis patterns for order and shipment of a product [4].

8 Discussion and Future Work

In this Section we want to highlight some of the limitation of the proposed approach as well as the benefits. The integration approach proposed in this paper may lead to a *traceability* problem. Once the two patterns are integrated it becomes hard, if not impossible, that we 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 work for small-scale projects, but for large 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.

Another limitation in the proposed approach is that it results in stable analysis pattern, and hence, it implicitly assumes that the developer is using software stability for her development, which may not be the case. This limitation calls for more general framework by which developer can decide the structure of the resultant integrated pattern. We currently investigate the visibility of such generic framework.

Nonetheless, the main advantage that we envision in our proposal is that we were able to virtually extend the stable analysis patterns reuse space to accommodate any existing analysis pattern. This fact expedites the development of stable analysis patterns, and hence, stable model, and allow for reusing other developer experiences in different fields. Consequently, we can avoid the current redundancy in analysis patterns (several analysis patterns address the same exact problem).

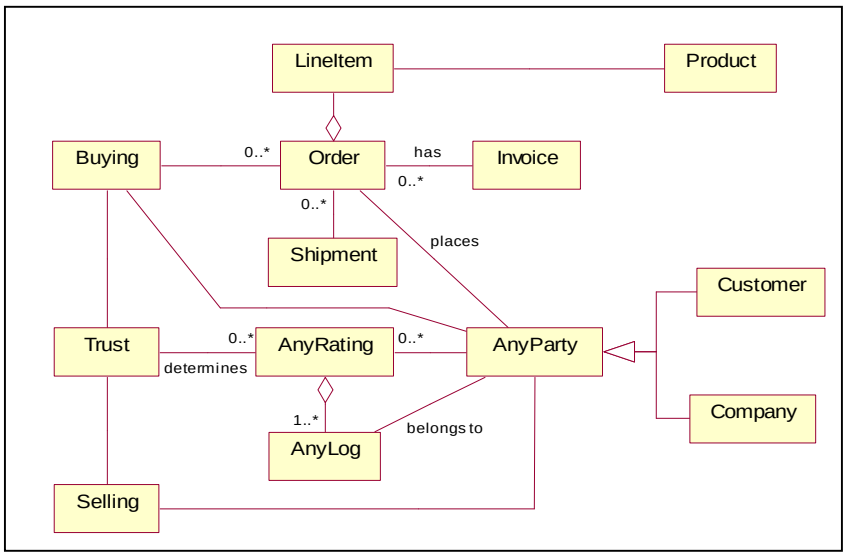


Fig. 9. The resultant integrated pattern of trusted order and shipment pattern

6 Conclusion

In this paper we address the problem of analysis pattern integration. We focus on the particular problem of integrating stable analysis patterns with traditional pattern to construct a new useful pattern, or more generally, a new analysis model. We define the problem as the *S-T Integration* problem and propose a generic iterative approach that consists of four-phase to solve this problem. We demonstrate the approach through two case studies. Our approach can help in reducing the duplicated effort in developing analysis patterns that address the same problem. In addition, the approach allows for virtually utilizing any available analysis patterns, and hence it stretches the reuse space of analysis patterns.

References

1. Schmidt D.C., Fayad M.E., and Johnson R.:Software Patterns. Communications of the ACM, Vol. 39, No. 10 (1996)
2. Hay D., : Data model pattterns-conventions of thoughts. Dorset House Publ, (1996)
3. Gamma, E., et al.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. Addison-Wesley Publishing Company, New York (1995)
4. Fernandez E.B., Yuan X., and Brey S.: Analysis patterns for the order and shipment of a product. In 7th Pattern Languages of Programs Conference, Monticello, IL, USA (2000)
5. Fernandez E.B., and Yuan X.: An analysis pattern for reservation and use of reusable entities. In 6th Pattern Languages of Programs Conference, Monticello, IL, USA (1999)
6. Fernandez E.B., and Yuan X.: "Semantic analysis pattern. In 19th Int. Conference on Conceptual," Modeling ER2000, pp. 183-195 (2000)
7. Hamza, H.: A Foundation for Building Stable Analysis Patterns. Master thesis, University of Nebraska-Lincoln, USA, August (2002)
8. Hamza, H.: Building Stable Analysis Patterns Using Software Stability. 4th European GCSE Young Researchers Workshop 2002 (GCSE/NoDE YRW 2002), Erfurt, Germany, October (2002)
9. Hamza H.: "A Framework for Software Analysis Patterns Integration," Technical Report, Computer Science and Engineering Dept. University of Nebraska-Lincoln, TR 04-04-04, USA (2004)
10. Hamza. H., Fayad, M.E.: Model-based Software Reuse Using Stable Analysis Patterns. 12th Model-based Software Reuse Workshop, 16th ECOOP '02, Malaga, Spain, June (2002)
11. Hamza. H., Fayad, M.E.: A Pattern Language for Building Stable Analysis Patterns. 9th Conference on Pattern Language of Programs (PLoP 02), Illinois, USA, September (2002)
12. Hamza. H., Fayad, M.E.: The Negotiation Analysis Pattern. In Proc. of the 11th Pattern Languages of Programs Conference, Monticello, IL, USA (2004)
13. Cline M., Girou M.: Enduring Business Themes. Communications of the ACM, Vol. 43, No. 5, May 2000, pp. 101-106
14. Fayad, M.E., Altman, A.: Introduction to Software Stability. Communications of the ACM, Vol. 44, No. 9, September (2001)
15. Fayad, M.E.: Accomplishing Software Stability. Communications of the ACM, Vol. 45, No. 1, January (2002)
16. Fayad, M.E.: How to Deal with Software Stability. Communications of the ACM, Vol. 45, No. 4, April (2002)

17. Fowler M.: Analysis Patterns: Reusable Object Patterns, Addison-Wesley (1997)
18. Coad P., North D., and Mayfield M.:Object models-strategies, patterns, and applications. Yourdon Press, Prentice-Hall, Inc. New Jersey (1995)
19. Braga R. T. Vaccare, et. al.:A Confederation of Patterns for Business resource Management. In Proc. of the 6th Pattern Languages of Programs Conference, Monticello, IL, USA (1998)
20. Yacoub S., and Ammar H.: Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems. Addison Wesley (2003)
21. Fayad M.E., and Hamza H.: The Trust Analysis Pattern. In the Proc. of 3rd Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP04) (2004).