JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 4 (April 2004)
Special issue: TOOLS USA 2003

Applying Analysis Patterns Through Analogy: Problems and Solutions

Haitham Hamza, University of Nebraska-Lincoln, U.S.A
Mohamed E.Fayad, PhD, San Josè State University, U.S.A.

Abstract

Traceability and generality are among the main qualities that determine the effectiveness of developed analysis patterns. However, satisfying both qualities at the same time is a real challenge. Most of the analysis patterns are thought of as templates, where they can be instantiated, and hence reused through an analogy between the original pattern and the problem in hand. Developing analysis patterns as templates might maintain the appropriate level of generality; however, it scarifies patterns' traceability once they are applied in the developed system. In this paper, we illustrate the main problems with developing analysis patterns as templates and reusing them through analogy. In addition, we demonstrate, through examples, how stable analysis patterns [Hamza, 2002a,Hamza and Fayad 2002a] can satisfy both the generality and traceability, and hence, enhance the role of analysis patterns in software development.

1 INTRODCUTION

In the last decade, patterns have emerged as a promising technique for improving the quality and reducing the cost and time of software development [Schmidt et al., 1996, Gamma et al., 1995]. A pattern can be generally defined as: *"An idea that has been useful in one practical context and will probably be useful in others"* [Fowler, 1997].

The obscurity of doing accurate analysis along with the fact that analysis is a tedious and time-consuming activity both makes the development of effective and reusable analysis artifacts of great interest. Analysis patterns form a promising base for facilitating and improving the quality of performing analysis. Some essential quality factors an analysis pattern should maintain in order to contribute effectively to the development process.

In this paper we focus on two of these qualities: *traceability* and *generality*. *Generality* means that the pattern that analyzes a specific problem can be successfully reused to analyze the same problem whenever it appears, even within different applications or across different domains. This quality factor is essential due to the fact

Cite this article as follows: Haitham Hamza, Mohamed E. Fayad: "Applying Analysis Patterns through Analogy: Problems and Solutions", in *Journal of Object Technology*, vol. 3, no. 4, April 2004, Special issue: TOOLS USA 2003, pp. 197-208.
http://www.jot.fm/issues/issue_2004_04/article11

that the analysis of a specific problem is the same if the problem remains the same. There is little sense in having different analysis models for the same exact problem. If analysis patterns fail to model the exact same problem when it appears in different applications, then the goal of developing patterns as reusable artifacts is diminished.

Traceability means that a pattern that is used in the development of a specific system can be successfully traced back to the original analysis pattern that has been used. Untraceable patterns will disappear once the developer instantiate them in their system, the fact that imposes further complications in the maintainability of the system.

Satisfying both generality and traceability is a factual challenge in current analysis patterns. This challenge is due to the fact that most current techniques for developing analysis patterns are based on viewing patterns as templates that form a general model for the problem. These templates can be reused through analogy [Fernandez and Yuan, 1999, Fernandez and Yuan, 2000, Fernandez, 2000, Vaccare et al., 1998]. As we will discuss in the following section, this approach may maintain patterns' generality to some extent; however, it may scarify their traceability.

In this paper, we illustrate the main problems in developing analysis patterns as templates and reusing them through analogy. As a remedy to these problems, we propose the use of the concept of *Stable Analysis Patterns* [Hamza, 2002a, Hamza, 2002b, Hamza and Fayad 2002a, Hamza and Fayad 2002b]. Stable analysis patterns are analysis patterns that are built based on the software stability concepts [Fayad and Altman 2002].

The paper is organized as follows: Section 2 describes the use of analysis patterns through analogy; Section 3 illustrates the problems associated with this approach; Section 4 provides an overview of stable analysis patterns; and Section 5 provides examples of using stable analysis patterns. The conclusions are presented in Section 6.

2 ANALYSIS PATTERNS AS TEMEPLETES

Most of analysis patterns are thought of as templates [Fernandez and Yuan, 1999, Fernandez and Yuan, 2000, Fernandez, 2000, Vaccare et al., 1998]. In [Coad et al., 1995], Code has defined patterns in general as follows: *"A pattern is a template of interacting objects, one that may be used again and again by analogy"*. That is, the pattern that is extracted from a specific project can be put into an appropriate abstract level such that it can be used to model the same problem in a wide range of applications and domains. The abstracted pattern is then considered to be a template, by which it could be used through an analogy. Developing patterns as templates, while providing an appropriate level of generality, it sacrifying their traceability when they are used by the means of analogy.

As an example of this approach, Figure 1 shows the class diagram of the *Resource Rental* pattern taken from [Vaccare et al., 1998], which forms the abstract template of the Resource Rental problem. The objective of the pattern is to provide a model that can be reused to model the problem of renting any resource; therefore, the class diagram does not tie the renting to a specific recourse. Figure 2 shows an example of using the

Resource Rental pattern in the application of the library service taken from [Vaccare et al., 1998]. Simply through an analogy, one can apply the original abstract pattern into a specific application.

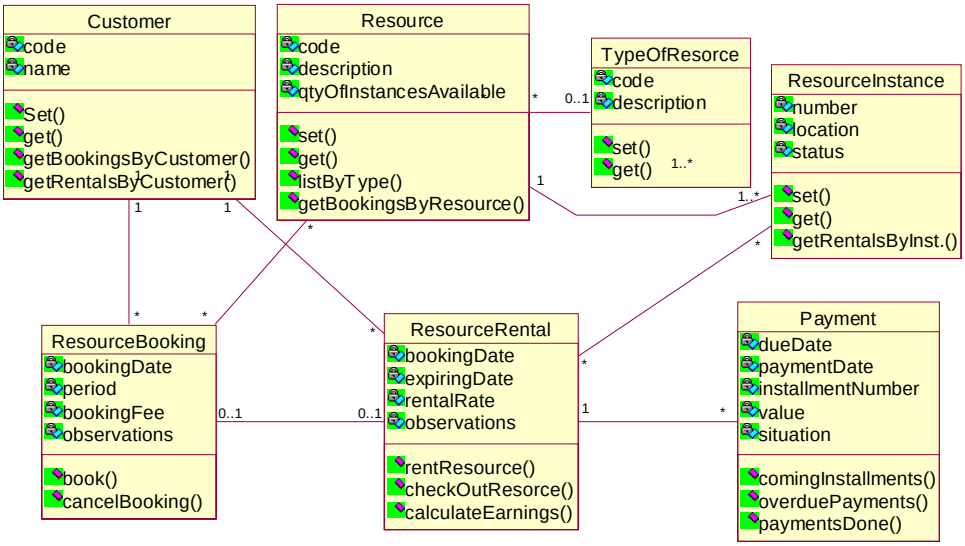


Fig. 1: Resource Rental pattern

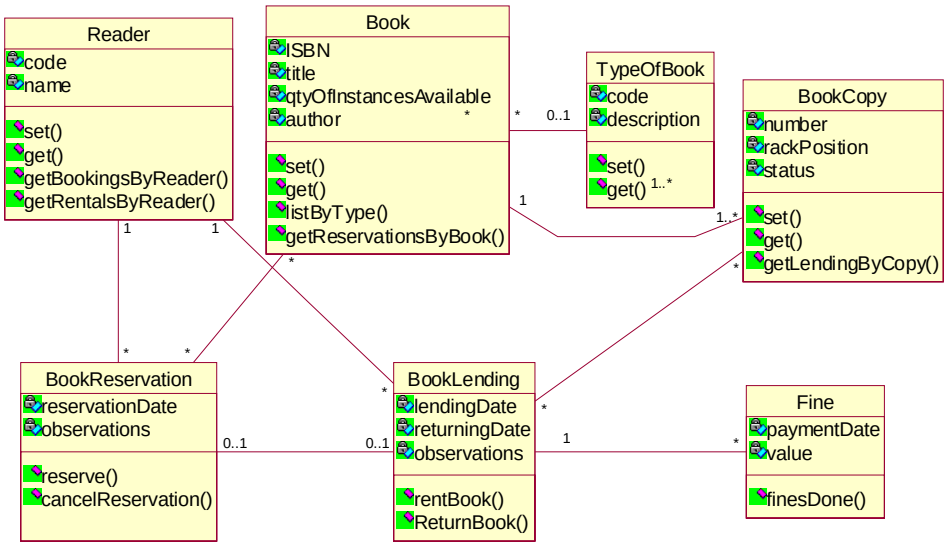


Fig. 2: Instantiation of Resource Rental pattern for a library service

3 PROBLEMS WITH USING ANALYSIS PATTERNS THROUGH ANALOGY

Though using analysis pattern through analogy might appear to be an appealing approach for maintaining a good level of pattern's generality; however, this technique raises some problems. Some of these problems are summarized below:

- ***Generate untraceable systems***. Once analysis patterns templates have been instantiated in the developed system, the original patterns are no longer extractable. For example, consider the instance of the Resource pattern shown in Figure 2 and imagine it as part of a complete library service system; it would be hard to extract the original pattern after such instantiation. This complication increases as the size of the developed system increases.
- ***Complicate system maintainability.*** Software maintenance is considered to be one of the most costly phases in the development life cycle. Therefore, complicating system maintainability is expected to further increase such cost. One can imagine a very simple situation where we need to update the developed system documentation due to some modification in system requirements. Since the developed system is using several patterns, identifying which patterns to be updated will be tedious and time consuming task.
- ***Trivialize classes' roles of the pattern*** To better discuss this issue we will use an example from [Fernandez and Yuan, 2000], where a class diagram for designing a computer repair shop is used, by an analogy, to build the class diagram of a hospital registration project. Thus, instead of shop that fixes broken computers we have a hospital that fixes sick people. We can simply replace the class named computer in the first project by the new class named patient in the next project. Even though such an analogy seems doable, it is impractical. There is a big difference between the computer as a machine and the patient as a human. These two classes might looks analogous since they both need to be fixed; however, their behaviors within the system are completely different. The role of the computer class is completely different from that of the patient. Therefore, such analogy is inaccurate. There would be even more differences if we try to generate the dynamic behavior of these two system using an analogy as suggested in [Vaccare et al., 1998].

4 STABLE ANALYSIS PATTERNS

Stable analysis patterns introduced in [Hamza, 2002a, Hamza, 2002b, Hamza, and Fayad 2002a], are analysis patterns constructed based on software stability concepts [Fayad and Altman 2002]. Before we describe how stable analysis patterns can satisfy both the generality and the traceability quality factors, a brief overview of software stability concepts, and an example of stable analysis patterns are provided in this section.

Software stability paradigm

Software stability stratifies the classes of the system into three layers: the Enduring Business Themes (EBTs) layer [Cline and Girou 2000, Fayad and Altman 2002], the Business Objects (BOs) layer, and the Industrial Objects (IOs) layer [Fayad and Altman 2002]. Based on its nature, each class in the system model is classified into one of these three layers. Figure 3 depicts the layout of the Software Stability Model (SSM) layers. Figure 4 shows the relationship between the different layers of SSM. The properties that characterize EBTs, BOs, and IOs are given in [Fayad 2002a, Fayad 2002b].

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 SSM. 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.

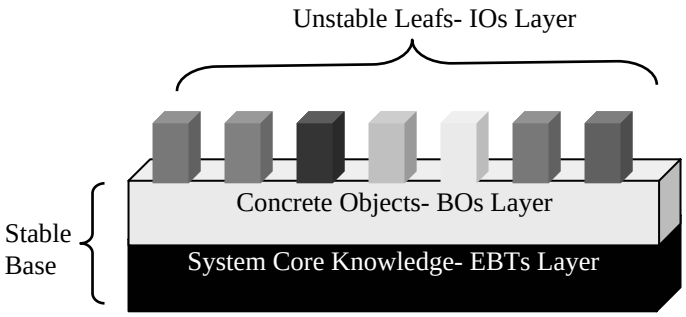


Fig. 3: SSM layers layout

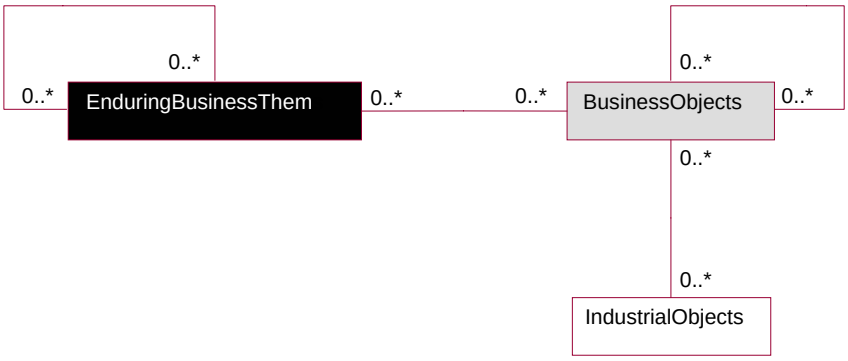


Fig. 4: The relation between SSM layers

Stable analysis pattern example

To illustrate the concept of stable analysis patterns we use a simple example. We use the *Negotiation* analysis pattern. *Negotiation* is a general concept that has many applications. In our daily life, there are various situations where negotiation usually takes place. For instance, buying or selling properties usually involves some sort of negotiation. In software systems, negotiation also appears frequently in the development of different applications. Developing software for online auctions and shopping might involve the negotiation of the price and/or the negotiation of different product aspects.

More technically, negotiation becomes an essential part in the development of next generation Web-based devices and appliances. Devices that need to access the Web diverge greatly in their capabilities, and hence negotiation mechanisms between client agent and the server play a fundamental role in deciding which representation of information a device should be given. Therefore, having a stable pattern that can model the basic aspects of a negotiation problem would make it easier for the developer to build their system by reusing and extending this pattern. Figure 4 shows the stable object model of the *Negotiation* pattern.

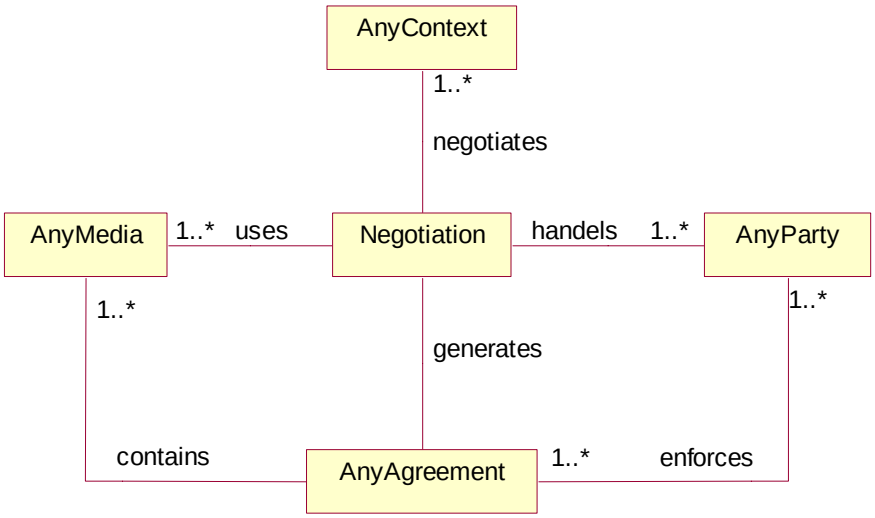
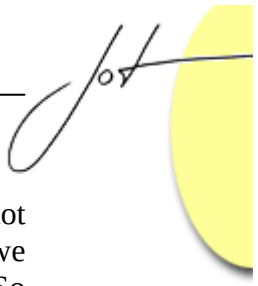


Fig. 5: *Negotiation* pattern stable object model

As shown in Figure 5 above, the *Negotiation* pattern consists of the following participants:

- ***Negotiation:*** Represents the negotiation process itself. This class contains the behaviors and attributes that regulate the actual negotiation process.
- ***AnyAgreement:*** Represents the result of the negotiation. The ultimate goal of any negotiation is to reach an agreement. Thus, this object presents a core element in any negotiation. It is important to note that in many cases negotiation ends with



no agreement and thus it is considered to be failed (the seller of the car did not agree on the price proposed by the buyer and vise versa), however, in this case we expect that the agreement should provide this result by whatever mechanism. So one can view the agreement object as the result of the negotiation, which is not necessary a successful result.

- ***AnyParty:*** Represents the negotiation handlers. It models all the parties that are involved in the negotiation process. Party can be a person, organization, or a group with specific orientation.
- ***AnyMedia:*** Represents the media through which the negotiation will take place. For instance, one can negotiate the price of a good over the phone. Others might use an email or a mail to negotiate specific issues in their business.
- ***Context:*** Represents the matters to be negotiated. If we are buying a home, many issues could be negotiated. For instance, the price of the home, the payment procedure, etc. Defining what is the issue to be negotiated is an essential element of any negotiation process, otherwise, negotiation will have no meaning.

The prefix 'any' that we used herein indicates that this is another pattern that provides an abstract model for the notion it precedes. For instance, AnyParty is a stand-alone stable pattern that models the party notation, and hence, can be used to model any party in any applications.

5 APPLYING STABLE ANALYSIS PATTERNS

In order to illustrate how stable analysis patterns can maintain both the generality and traceability quality factors, we use the Negotation pattern to model two different applications: Negotiation of buying a car, and Content Negotiation using Composite Capability/ Preference Profile (CC/PP). For simplicity, we give parts of the models in both examples that help to demonstrate the usage of the proposed pattern, and hence, these models are not complete. The full analysis (CRC- cards, use case diagrams, use case descriptions, sequence diagrams, and state transition diagrams) of these two examples is given in [Hamza, 2002a]. In the models given in Figures 6 and 8, we use the black color to denote the EBT objects, gray color to denote BO objects, and while color for IO objects.

Example 1: Negotiation to buy a car

In buying a car, a negotiation concerning the car's price and warranty usually takes place. This example models the simple negotiation that might be involved in buying a car. Figure 6 shows the stability model of the negotiation used in buying a car.

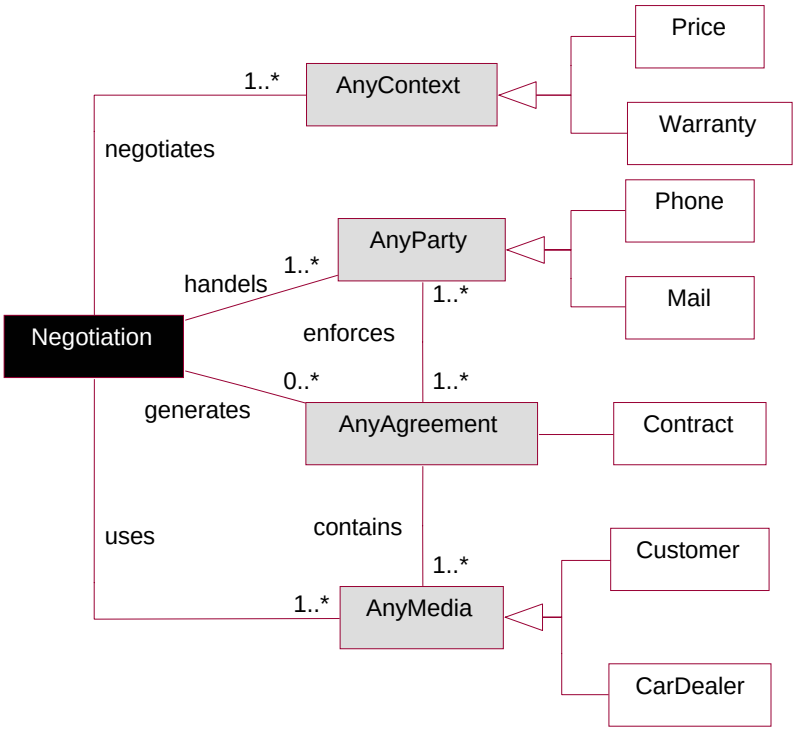


Fig. 6: Stability model of the negotiation in buying a car example

Example 2: Content Negotiation using Composite Capability/Preference Profile (CC/PP)

Today, very heterogeneous devices are required to access the World Wide Web; yet, each device has its own set of capabilities. As a result, a negotiation between the client and the server should take place in order for the server to know the capabilities of these devices and provide the appropriate contents. One possible techniques of performing content negotiation is called *Composite Capability/Preference Profile* (CC/PP) [W3C 2000]. A possible scenario of CC/PP content negotiation is given in Figure 7. Figure 8 shows the stability model of this example. Again, classes that are not in the original *Negotiation* pattern are colored in gray.

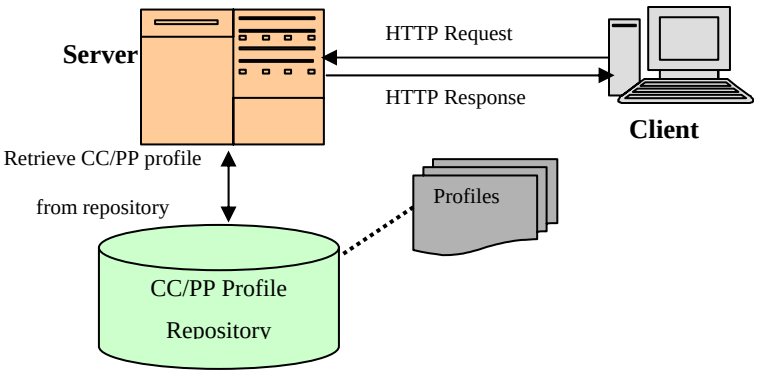


Fig. 7: Possible scenario of content negotiation using CC/PP

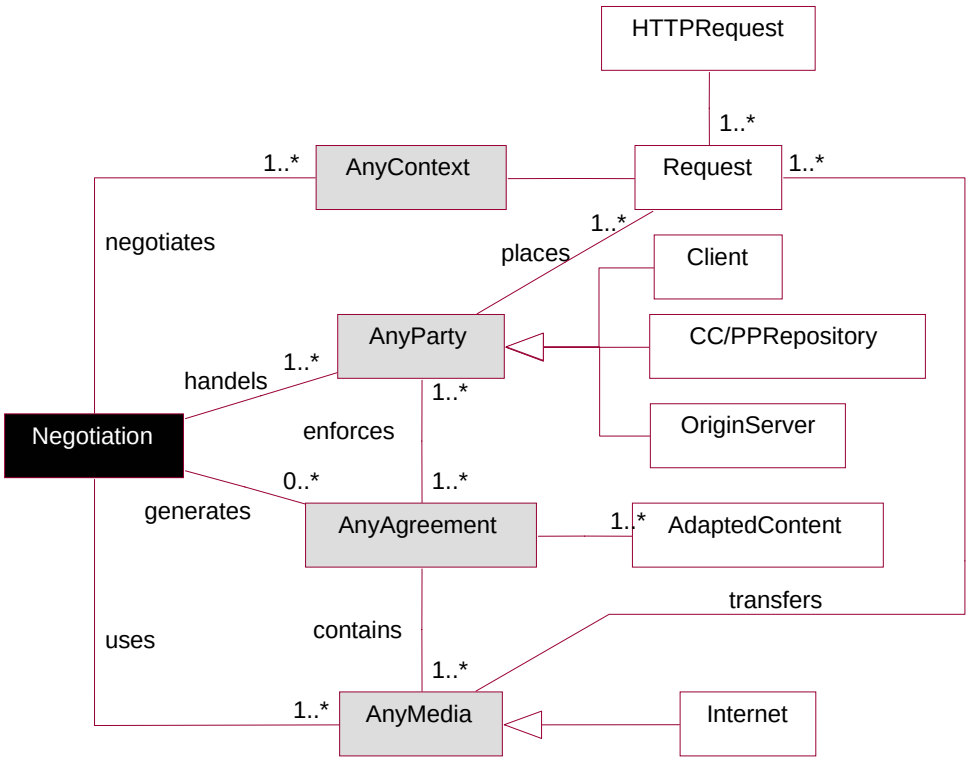


Fig. 8: The stability model of the content negotiation example

As shown in the two examples, the use of the *Negotiation* pattern is not achieved through an analogy. The pattern can be spotted easily and thus it is possible to trace it back in the developed system. On the other hand, the developed pattern does not lose its generality,

as we were able to apply it to model the same problem in two different applications. Moreover, one can realize that each object in the *Negotiation* pattern has a clear role independent of the application the pattern will be used in. For instance, AnyMedia as an object still exists and it has the same role independent of the application; however, the type of media might vary based on the application.

6 CONCLUSIONS

Stable analysis patterns introduce a new vision of developing and utilizing analysis patterns in building software systems. Although current approaches of developing analysis patterns as templates and utilizing them through an analogy maintain pattern generality; it scarifies its traceability. This makes the developed systems harder and more costly to maintain. Stable analysis patterns are developed and utilized so that they can preserve both the generality and traceability. In addition, stable analysis patterns guarantee the preservation of the classes' roles within the pattern; thus, each class has the same role independent of the application that the pattern will be deployed in. Therefore, stable analysis patterns can form a more effective base for utilizing patterns in developing software systems.

REFERENCES

[Cline and Girou 2000]

Cline, M., and Girou, M. "Enduring Business Themes". *Communications of the ACM*, Vol. 43, No. 5, pp. 101-106, 2000.

[Coad, et al. 1995]

Coad, P., North, D., and Mayfield, M. "Object models-strategies, patterns, and applications". *Yourdon Press*, Prentice-Hall, Inc. New Jersey, 1995.

[Fayad 2002a] Fayad, M. E. "Accomplishing software stability". *Communications of the ACM*, Vol. 45, No. 1, 2002.

[Fayad 2002b] Fayad, M. E. "How to deal with software stability". *Communications of the ACM*, Vol. 45, No. 4, 2002.

[Fayad and Altman 2002]

Fayad, M. E., and Altman, A. "Introduction to software stability". *Communications of the ACM*, Vol. 44, No. 9, 2002.

[Fernandez, 2000]

Fernandez, E. B. "Stock Manager: An analysis pattern for inventories". In *7th Pattern Languages of Programs Conference* (PLoP'2k), Monticello, IL, USA, 2000.

[Fernandez and Yuan, 1999]
Fernandez, E. B., and Yuan, X. “An analysis pattern for reservation and use of reusable entities”. In *6th Pattern Languages of Programs Conference* (PLoP’99), Monticello, IL, USA, 1999.

[Fernandez and Yuan, 2000]
Fernandez, E. B., and Yuan, X. “Semantic analysis pattern”. In *19th Int. Conference on Conceptual Modeling* ER2000, pp. 183-195, 2000.

[Fowler 1997] Fowler, M. *Analysis patterns: reusable object models*. Addison-Wesley, 1997.

[Gamma et al., 1995]
Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesely, 1995.

[Hamza, 2002a]
Hamza, H. “A foundation for building stable analysis patterns”. Master Thesis. University of Nebraska, Lincoln, USA, 2002.

[Hamza, 2002b]
Hamza, H. “Building stable analysis patterns using software stability”. *4th European GCSE Young Researchers Workshop* (GCSE/NODE), Erfurt, Germany, 2002.

[Hamza and Fayad 2002a]
Hamza, H., and Fayad, M. E. “Model-based software reuse using stable analysis patterns”. In *12th Workshop on Model-based Software Reuse, 16th ECOOP 02*”, Malaga, Spain, 2002.

[Hamza and Fayad 2002b]
Hamza, H., and Fayad, M. E. “A pattern language for building stable analysis patterns”. In *9th Pattern Languages of Programs Conference* (PLoP’02), Monticello, IL, USA, 2002.

[Hay, 1996] Hay, D. *Data model pattterns-conventions of thoughts*. Dorset House Publ., 1996.

[Schmidt et al., 1996]
Schmidt, D. C., Fayad, M. E., and Johnson, R. “Software Patterns”. *Communications of the ACM*, Vol. 39, No. 10., 1996.

[Vaccare, et al. 1998]
Braga, R. T. V., Germano, F. S. R., and Masiero, P. C. “A confederation of patterns for business resource management”. In *5th Pattern Languages of Programs Conference* (PLoP’98), Monticello, IL, USA, 1998.

[W3C 2000] Composite Capability/Preference Profiles (CC/PP): “A user side framework for content negotiation”. W3C Note 21 July 2000. http://www.w3.org/TR/NOTE-CC

About the authors

Haitham Hamza is a Ph.D. student at the University of Nebraska-Lincoln. He can be reached at hhamza@cse.unl.edu

Mohamed E. Fayad is a Full Professor of Computer Engineering at Josè State University http://www.engr.sjsu.edu/fayad. He can be reached at m.fayad@sjsu.edu or fayad@activeframeworks.com