# Aspects in Communications: Performance

M. E. Fayad
Computer Engineering Dept.,
College of Engineering
San Jose State University
One Washington Square, San Jose,
CA 95192-0180
(408) 924-7364
m.fayad@sjsu.edu

R. S. Pradeep
Computer Engineering Dept.,
College of Engineering
San Jose State University
One Washington Square, San Jose,
CA 95192-0180
(408) 247-4575
rohinisp@hotmail.com

F. Seddiqui
Computer Engineering Dept.,
College of Engineering
San Jose State University
One Washington Square, San Jose,
CA 95192-0180
(408) 296-6590
iqra@pacbell.net

## ABSTRACT
This paper deals with the similarities between "Aspect Oriented modeling" and "Software Stability modeling" techniques. The design pattern for performance is considered to elaborate on these issues. "Aspects", which might arise at any stage of the software lifecycle, have attributes that are similar to the "Enduring Business Themes" and "Business Objects" used in the stable modeling.

## 1. INTRODUCTION
Aspect oriented modeling is a modeling technique that attempts to abstract out features common to many parts of the system beyond functional modules, thus improving the quality of modeling [4,5]. Like objects, aspects may arise at any stage of the software lifecycle, including requirements specification, design, implementation, etc.

With the present systems and languages, the design for the abstraction mechanisms, that includes defining and obtaining aspects, involves breaking a system down into parameterized components that can be called to perform a function. But many systems have properties that don't necessarily align with the system's functional components. The examples of such cases are usually communication, failure handling, coordination, memory management, or real-time constraints, that tend to cut across groups of functional components. The issues these design decisions address are "aspects". Aspects are usually used for improving the separation of goals and concerns in software design and implementation.

On the other hand, Stable Modeling is a modeling technique that uses the concepts of "enduring business themes" (EBTs) and "business objects" (BOs) [1, 2]. These form the core of the system that is modeled. Hence, this reduces the changes that need to be made to the system when the requirements change. To obtain EBTs and BOs of the system under consideration, those aspects of the system that remain stable over a period of time are identified. These usually require minimal reengineering and code modifications when requirements change. The majority of engineering done on a system modeled should be to fit the project to those areas that remain stable. This yields a stable core design and thus, a stable software product. Changes introduced to the software product will then be in the periphery since the core would be based on something that remains stable. Since any changes that must be made to the software in the future will be in this periphery, it will only be these small external modules that will be engineered. Thus, the endless circle of reengineering the entire system is avoided for any minor changes.

Let us examine how aspects have properties similar to EBTs and BOs considering a performance design pattern. There are several methodologies available for developing patterns. Unfortunately, developing a good pattern with an appropriate methodology is expensive and has drawbacks. The pattern that is developed may not necessarily be applicable to multiple domains. This paper considers a performance design pattern developed using Software Stability Model [1, 2] and the concept of Stability Patterns [3]. The pattern is simple to learn and provides enough features to hook to applications with minimal modifications. The pattern for performance is dealt with in the following sections. We provide the problem and the solution along with the applicability. This pattern is also used as an example to show the similarities in properties of aspects, EBTs, and BOs.

## 2. PATTERN EXAMPLE FOR STABLE DESIGN MODELING

### 2.1 Pattern Name
***AnyPerformance:*** This name indicates that the pattern ***AnyPerformance*** is the process of accomplishing a task in accordance with a set standard of accuracy and completeness. Since it begins with "Any", it indicates that this can be used by any domain that has any kind of performance involved.

### 2.2 Problem
Since the pattern ***AnyPerformance*** spans many contexts that are completely different in nature, modeling a generic concept that can be applied to all domains is the problem at hand. This is due to the fact that the requirements differ based on the domain or the context.

The performance of the system or a party can be different based on the requirements, objectives, and measures. For example, objectives for performing a task for wireless networks will be different from that of a performer in a theatre, or an employee at work. Hence, obtaining a generic model or a pattern in this case that encompasses all the features of different domains can be a difficult task to accomplish. How a single model addresses these

variations is the challenge faced by this model. This leads to an area that requires a solution to the problem of how a model that handles performance for different applications be obtained. But there are certain aspects of performance that transcend all application domains, which form the participants of the pattern in the form of EBTs and BOs, These have been identified and described in the solution section for the pattern.

## 2.3 Context

Performance is an important concept in any domain that needs its party to perform some task according to the standards desired. A reason as to why the performance task needs to be carried out may also be specified to the performer along with the criterion for the desired performance. In general, this term may be either a criterion objective or an enabling objective. For example, the objective for an actor to perform on stage may be to entertain people, or for a network, the objective for performing a particular task may be specified by the domain based on the type of the services desired. There might be some actions or data that can be objectively observed, collected, and measured to determine if a task performer has performed the task to the prescribed standard or that which can be used for further analysis and evaluation.

In some aspects, like telecommunication networks, performance is an important concept for carrying out the task of delivering data and voice in a wireless mode according to the set standard. There can be other aspects, like specifying the channel allocation strategy, protocols, architecture, or security features that form the requirements for a particular performance. Performing a task will also be required in industries that are involved in assembling or manufacturing. Therefore, evaluating performance across domains becomes easier using a stable pattern. This pattern can be reused with many applications by using simple hooks that require minimal changes without the entire system being rewritten.

## 2.4 Forces

The design pattern obtained for performance spans many contexts that are completely different in nature. Performance of any task can be done by one or more entities simultaneously that can be requested by multiple parties. Thus, the pattern needs to handle multiple parties and entities. How these multiple entities and parties are handled is a challenge faced by this model. The pattern is also not flexible enough to address the requirements of domain-specific features. Criteria can be different based on the context it occurs in. The features for criteria, defined by a party, are domain specific and addressing these features is a limitation faced by this model.

## 2.5 Solution

The solution that is proposed concentrates on obtaining a generic pattern for performance that can be used with any domain, leaving out the domain specific features. This allows any application to be hooked to the pattern with a few changes.

Figure1 below shows the diagram of the ***AnyPerformance*** pattern

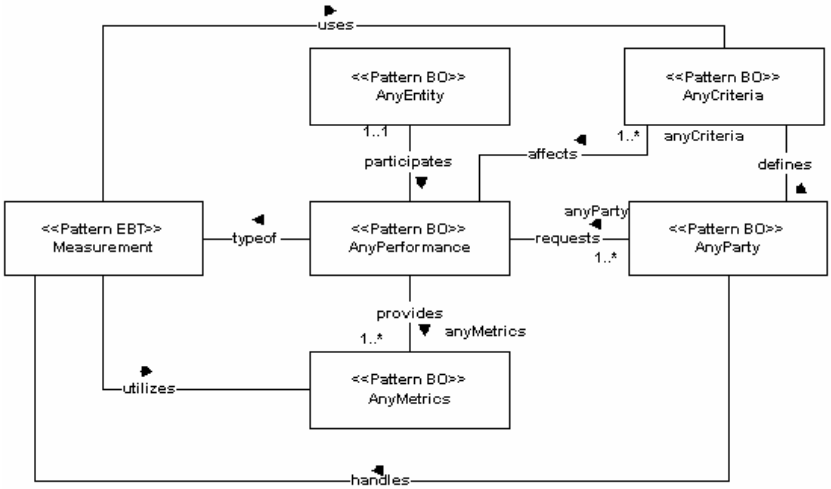


**Figure 1: Performance Design Pattern**

The pattern ***AnyPerformance*** consists of the following participants:

**Classes**

- ***AnyPerformance***: This is the core of the stable design mode and represents the process of accomplishing a task in accordance with a set standard of accuracy and completeness.

**Patterns:**

- ***Measurement:*** This represents the EBT from which the business object performance is derived. It utilizes the data and the observations that are obtained as a result of some task being performed and checks if the task is performed to the set standard which is provided by the BO criteria.
- ***AnyParty***: This represents parties that are involved in the task of performance directly or indirectly. They can be involved in the process of requesting a task to be performed or in the process of observation and providing measures. For example, the party can be an employer who requests the performance or some task to be performed to make relevant observations and data collections or it can also be an audience in a theater. In more technical aspects, it can be an operator requesting a network to transmit data.
- ***AnyEntity:*** This represents any entity that needs to perform a task. It can be a wireless system, an engine part, etc., which needs to finish or perform a task that is assigned to it. The engines may perform the task of running some machinery in an industry or a network, which can be wireless or wired, may perform the task of transmitting data.
- ***AnyCriteria:*** This provides the details of all those issues that affect performance. It can be a standard to be achieved, a condition to be satisfied, or a reason for performance to be carried out as a task. Its also specifies a criterion for the desired performance by the performer. In general, this term may either be a criterion or an enabling objective. It also represents all separate acts or things that are required to satisfactorily complete any party's performance on the job. It includes the act (behavior), the conditions under which the behavior is performed, and the standard of performance required by the incumbent.
- ***AnyMetrics:*** Describes the actions and data that can be objectively observed, collected, and measured to determine if

a task that a performer has performed is to the prescribed standard. In general, this represents all the assessment data that can be collected, or the observations that can be made after the completion of a performance. This metrics can be further used for analysis and evaluation.

## 2.6 CRC Cards

The CRC cards give details on the responsibilities and collaborations for each class in the pattern

**Table 1: CRC Card for AnyPerformance**

| AnyPerformance ( Performance facilitator) | | |
|---|---|---|
| Responsibility | Collaboration | |
| Defines the process of accomplishing a task in accordance with a set standard of accuracy and completeness | Client | Services |
| | AnyParty<br>AnyEntity<br>AnyCriteria<br>Measurement<br>AnyMetrics | definePerformance()<br>analyzeEntityPerformance()<br>performanceResult()<br>calculatePerformance()<br>criteria ()<br>metrics () |

**Table 2: CRC Card for AnyEntity**

| AnyEntity (Performer) | | |
|---|---|---|
| Responsibility | Collaboration | |
| Performs a task to a standard set | Client | Services |
| | AnyPerformance | performTask()<br>providePerformanceData() |

**Table 3: CRC Card for AnyParty**

| AnyParty (Performance Requestor) | | |
|---|---|---|
| Responsibility | Collaboration | |
| Setting the rules under which the performance of any entity is measured | Client | Services |
| | AnyPerformance<br>AnyCriteria | evaluateEntityPerformance ( )<br>defineCriteria( ) |

**Table 4: CRC Card for AnyCriteria**

| AnyCriteria (Criteria Definer) | | |
|---|---|---|
| Responsibility | Collaboration | |
| Describes the rules, standards and the external factors that need to be considered before any task can be performed | Client | Server |
| | AnyPerformance<br>Measurement<br>AnyParty | defineCriteria()<br>modifyCrietra() |

**Table 5: CRC Card for Measurement**

| Measurement (Measurement) | | |
|---|---|---|
| Responsibility | Collaboration | |
| Utilizes the data and the observations obtained as a result of some task being performed to check if the task is performed to the set standard | Client | Server |
| | AnyPerformance<br>AnyMetrics<br>AnyCritera | specifyMeasurement()<br>compareMetricsResults() |

**Table 6: CRC Card for AnyMetrics**

| AnyMetrics(Metric Provider) | | |
|---|---|---|
| Responsibility | Collaboration | |
| Describes the actions and data that can be objectively observed, collected, and measured to determine if a task that a performer has performed is to the prescribed standard | Client | Server |
| | AnyPerformance<br>AnyEntity<br>Measurement | provideMetrics ( ) |

## 2.7 Consequences

Performance pattern supports its objectives in the following ways:

- Generalized process for performance applicable to various domains with sufficient flexibility is obtained.
- Scalable pattern in terms of parties and entities in various domains that use performance is derived.

This pattern has the following benefits:*Adaptable for different kind of entities*: The performance pattern has high level of adaptability making it possible to adapt this pattern for different kinds of entities. For example, from base standards like the IEEE 802.3 and IEEE 802.11, different protocols are developed. This performance pattern is used to check the efficiency of all the protocols developed from these standards.

- *Easy to use*: The performance pattern developed is a general pattern that can be adapted for different parties and metrics. This provides a high level of extensibility to handle adding new and complex features to this model.

## 2.8 Tradeoffs

Performance pattern has following tradeoffs:

- *Generalization*: Generalization across domains is obtained at the cost of flexibility.
- *Feature extraction*: Different features exist for different types of domains and contexts, i.e., the features for industries will be different from the features for wireless networks and hence it is difficult to extract the common features and make them applicable in this pattern in an in-depth manner.

## 2.9 Results

The results obtained from the pattern are listed below:

- Obtained a generic pattern for performance evaluation applicable across various domains.
- Obtained a stable, reusable pattern for performance, to which various applications can be hooked.
- Obtained a scalable pattern, in terms of entities, metrics, participants, and media

## 2.10 Applicability with Illustrated Examples

### *2.10.1 Problem Description*

The pattern can be used for performance of the task of transmission of data in networks.

### *2.10.2 Problem Description*

The diagram below depicts the class diagram for the scenario of data transmission in networks. The stable parts of the system, i.e., the EBTs and BOs are shown in different columns of the table.

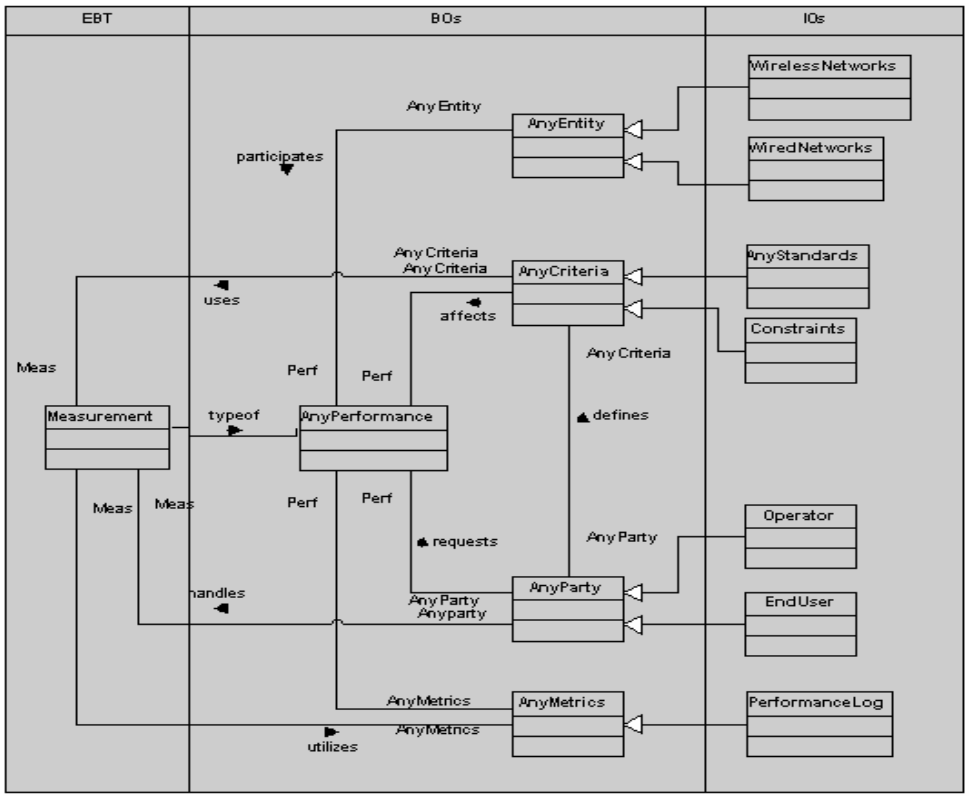


**Figure 2: Stability model class diagram for Performance**

Let us now examine how this example helps in evaluating the similarities between the aspects, EBTs and BOs.

# 3. COMPARISON OF ASPECT, EBT AND BO

This section provides information on various properties of aspects and compares it with the EBTs and BOs from stable modeling. Figure 1 gives the design pattern, obtained using stable modeling technique for performance of any task, in any system, by any party or entity, in any domain. Figure 2 describes how this pattern can be applied to one of the domains that use performance.

Using AOAD (Aspect Oriented Analysis and Design), has many advantages. The following list illustrates the properties of aspects and how they are similar to the EBTs and the BOs obtained by stable modeling.

- ***Stable:*** Aspect modules can usually be unaware of the crosscutting concerns, and hence it is easier to add new functionalities and introduce new aspects without altering the system. Furthermore, when new modules and aspects are added, existing aspects crosscut them, creating a coherent evolution. As a result, the system remains stable over a period of time handling new requirements without requiring the entire system to be redesigned. Similarly, with the EBTs forming the core of any model, which identifies those aspects of the system that will not change and still remain flexible to handle future requirements, the model developed, remains stable. The part of the system that is likely to change over a period of time forms the periphery of the model in the form of "industrial objects". Thus, the property of stability can be applied to aspects, EBTs and BOs.
- ***Reusable:*** Aspects can be easily reused since each aspect forms a separate, individual module. Similarly, each EBT along with its related BOs can be reused in any application that uses the concept mentioned by the EBT. For e.g., in figure.1, any application using performance can reuse the BO "***AnyPerformance***", as a design pattern with minor modifications done to hook required applications.
- ***Domain independent:*** Aspects are domain independent. They provide solutions that can be applied to multiple domains, by providing domain-specific mechanisms to solve a particular problem. Cross-cutting concerns are addressed in a modularized way. Figure.1 depicts how stability modeling, when applied to a problem, addresses the issue in a domain independent way. The pattern for performance obtained is general enough to be applied to any domain that uses performance.
- ***Patterns:*** Since most of the systems developed have cross cutting concerns that result in code tangling and code scattering, a few new techniques that handle modularization have evolved to handle these issues. Patterns are one such technique that enables deferring implementation. It also provides the features of modularization and code optimization. Aspect oriented modeling and stable modeling

support analysis and design patterns that are general enough to address and handle future requirements. Since a pattern needs to be reusable, avoiding the redesign of the entire system, it becomes important to develop a good pattern that can be applicable to multiple applications and domains. Figure.1 gives an example of a pattern developed using stable modeling techniques for performance in which the BO "AnyPerformance", remains stable changing only internally.

- ***Dynamic:*** Aspects are dynamic, which means that the properties of an aspect can change over a period of time internally to encompass the changing requirements. Similarly, the BOs in a stable model are dynamic, changing internally to handle any changes that are done to the system, but remain stable externally, keeping the model stable. This is the reason why they are partially tangible and will remain externally stable throughout the existence of the problem.
- ***Handles goals and purposes***: An aspect or a concern, handles a particular concept, goal, or area of interest. EBTs, in a similar way, focus on the goal of developing the system. This concept helps in obtaining the answer to the question of why the system under consideration is being modeled. In the example considered in figure.1, the question of why performance is done is answered in the form of the EBT, "Measurement", which states that performance is a type of measurement. The BO "AnyMetrics", provides the result of any performance as a measurement. Hence, the purpose of performing a task by any party or entity in this case is to obtain measurement for that performance which can be used for further analysis.
- ***Quality***: Aspects are modeled in such a way that they avoid poor code quality when implemented. This is accomplished by keeping away hidden problems caused due to code tangling. It also handles issues related to targeting too many concerns at once, which results in one or more concerns receiving less attention and thus, resulting in a poor concern or aspect. Quality is assured in a similar way when EBTs and BOs are used in modeling. The layering of EBTs and BOs, in the system being modeled, along with the IOs forming the peripheral, takes care of this issue.

# 4. CONCLUSION

In conclusion, aspects in aspect oriented modeling and EBTs, along with the BOs, in the stable modeling, present a good system design and architecture that considers the present and the future requirements. This ultimately avoids a patchy-looking implementation due to a number of problems related to system design rework, reuse, quality, and stability. The similarities in properties mentioned earlier provide evidence of the similarities in features between aspects, EBTs, and BOs.

# 5. REFERENCES


[1] M.E. Fayad, and A. Altman. "Introduction to Software Stability". Communications of the ACM, Vo.44, No. 9, September 2001, pp 95-98.

[2] M.E Fayad. "Accomplishing Software Stability." Communications of the ACM, Vol.45, No. 1, January 2002.

[3] H. Hamza and M.E. Fayad. "A Pattern Language for Building Stable Analysis Patterns", 9th Conferenceon Pattern Language of Programs (PLoP 02), Illinois, USA, September 2002.

[4] Ramnivas Laddad. "I want my AOP", retrieved on July 15, 2003. http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect_p.html

[5] Gregor Kiczales, John Lamping, Anurag Mandhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin. "Aspect-Oriented Programming". European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlang LNCS 1241. June 1997

[6] R.S. Pradeep and M.E. Fayad. "The Pattern Language for Performance Evaluation of Wireless Networks" In Progress.