Pattern language for performance evaluation

Rohini Pradeep
San Jose State University

PATTERN LANGUAGE FOR PERFORMANCE EVALUATION

A Thesis

Presented to

The Faculty of Computer Engineering

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Rohini Pradeep

August 2004

APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING

[illegible]

Dr. Mohamed. E. Fayad

[illegible]

Dr. Rod Fatoohi

[illegible]

Dr. Ammar Rayes, Cisco Systems, Inc.

APPROVED FOR THE UNIVERSITY

[illegible]

ABSTRACT

PATTERN LANGUAGE FOR PERFORMANCE EVALUATION

by Rohini Pradeep

This thesis describes Software Stability modeling technique along with related analysis and design patterns. It mainly addresses the pattern language for Performance Evaluation and explains how this language can be used to obtain the relationship between certain stable patterns. These patterns help overcome the issues related to "forgettable" or "disposable" systems that result in redesigning an entire system to handle new functionalities. Pattern Language for Performance Evaluation provides sets of patterns that address the core system behavior. These patterns exhibit a rich set of behaviors capable of handling features that are reusable, adaptable, and maintainable, with minimal modifications.

The contribution to the pattern language field includes a Stable Analysis Pattern, ***Evaluation***, Stable Design Pattern, ***AnyPerformance***, Stable Architectural Pattern, ***Performance Evaluation***, and a Pattern Language Map for Performance Evaluation. Several application scenarios for these patterns are also implemented and discussed in detail.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

1.1 Introduction

While developing software systems may be complex, maintaining software and enhancing it with requirements and making it applicable to multiple domains can be a far greater challenge. Designing reusable, object-oriented software that remains stable, even after addressing new requirements, is a complicated issue. Some applications are designed, implemented, maintained, and sparingly used, while also exhibiting limited purpose and short life span. Because they are difficult to maintain, expensive to reuse, modify, or adapt to new requirements, these systems are usually termed "forgettable" or "disposable." Consequently, such systems are discarded and replaced by entirely new systems.

The goal of the thesis, Pattern Language for Performance Evaluation, is to overcome these issues. The proposed solution is not intended to replace existing systems, rather, the effort is to define and provide a new methodology for designing such systems with a rich set of patterns for Performance Evaluation that can be used efficiently. Pattern language is intended to provide a set of highly compatible, related patterns, which will help increase the overall flexibility of the system under consideration. It provides a generic set of patterns pool for Performance Evaluation, from which any set of patterns can be chosen to suit an application. Since these patterns are based on stability modeling approach, which focuses on the stability of these patterns and hence the application built from these patterns, the application developed is usually not "disposable" or "forgettable." The details of this approach and how this modeling technique helps overcome the issues of "disposable" systems are discussed in detail in the later chapters. These start by providing an introduction to the pattern language for Performance Evaluation and the modeling technique used for obtaining these patterns. The next few chapters present the organization of the patterns in pattern language and how this interrelationship deals with issues arising from unpredictable future requirements and enhancements. The following sections address the research methodology; how this thesis is conducted and what significant contribution it has made to the software engineering and patterns community.

1.2 Research Methodology Overview

Pattern language is all about providing a set of patterns with their problems and solutions. In simple terms, it enables us to document a set of known recurring problems along with their solutions in any context and encourage a user to pick out the relevant patterns suitable for their application. Hence, it becomes important to understand the goals and missions of the system, along with its capabilities, architectures, etc. Figure 1 provides a detailed picture of what the user needs from three different perspectives:

1. Marketing or business language
2. Stability
3. Scientific research

The marketing or the business language is mainly applicable and used across all industry sectors and domains of electronic or non-electronic trade. The pattern language in terms of business language provides a formal description of a specific business terminology that can be used to identify uniquely the corresponding terms in stability and research. The stability language, on the other hand is a systematic means of communicating the ideas of stability methodology for patterns language. And finally, research mainly identifies the search for knowledge, which involves systematic investigation to establish facts about pattern language for performance evaluation.

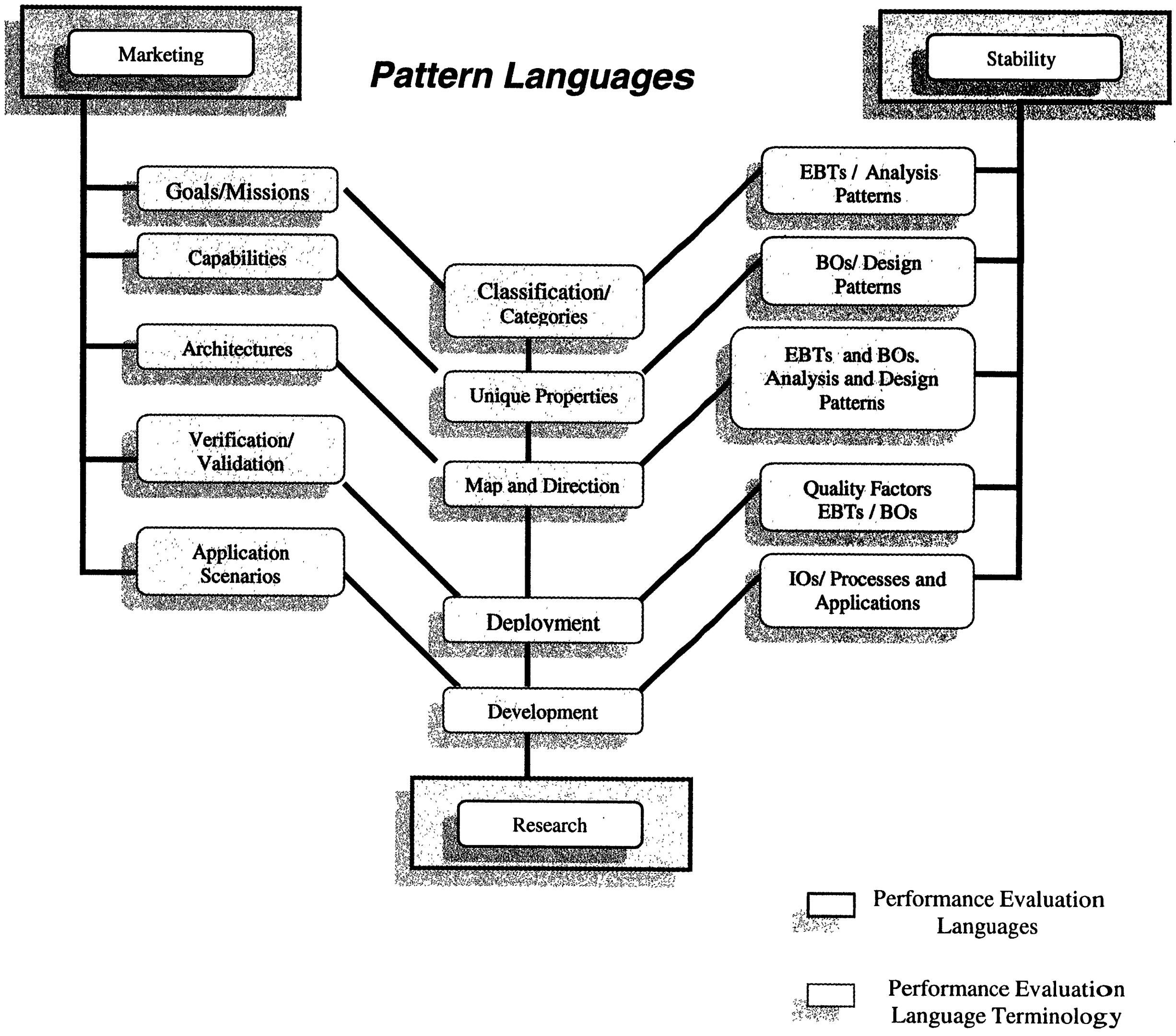


Figure 1: Pattern Language

As shown in the Figure 1, the pattern language for performance evaluation in the business perspective addresses the goals or capabilities of the system, which relates to the Enduring Business Themes (EBTs) or analysis patterns in stability and categories/classification in research. This addresses the goal of the system, which helps identify those aspects that remain consistent throughout the existence of the system.

It is a challenging endeavor to identify these aspects because they have to remain the same no matter what the application is. Similarly, capabilities in terms of business language map onto unique or common properties in stability, which encompass all the properties of the goal identified. These form the Business Objects (BOs) or the design patterns, which occur in the solution space and help to uniquely identify the goals. The architecture, which is usually a combination of goals with capabilities, provides the map and direction of Performance Evaluation. In other words, it consists of both the analysis and design patterns fulfilling a specific functionality with the domain of Performance Evaluation.

Verification and validation, which occurs next in the business language deals with a board diversity of quality factors that need to be handled during deployment phase in stability. These are again several design and analysis patterns for quality assurance. The application scenarios provide the details on applicability of the patterns identified in a plethora of applications and are termed development in research.

1.3 Thesis Methodology

The focus in this section is to provide a brief description of the methodology used in the development of the pattern language and its associated patterns. The methodology for this thesis is based on a wide approach to accessing and obtaining the relationship between several interrelated patterns for performance evaluation. It can be broadly viewed as consisting of three logical stages. The first stage involves identifying the related patterns in the pattern language. The second stage then involves categorizing them as either analysis patterns or design patterns based on the whether the pattern lies in the problem space or the solution space. The third phase aims at providing the relationship between different patterns and implementing them in different applications. These are discussed in detail in different chapters.

The first phase, which involves obtaining the context in which pattern language and the relevant patterns exist, made this thesis an interactive and feedback based study. This report aims at encompassing different patterns obtained based on direct subject and application observation, often diverse in nature. The method utilized to collect different aspects involved a series of questions that required some research. This involved discussion, particularly with the thesis advisor, Dr. M. E. Fayad, and the Stability Patterns Group. The team focused on the following points:

1. The goal of designing the patterns, or in other words, the reason for the pattern language to be developed

2. The different aspects or characteristics existing in different applications, which try achieving this goal
3. Interrelationship between the patterns identified
4. Possible application scenarios where these patterns can be implemented

1.4 Contribution

The contribution of the thesis is significant to the research community in the field of software engineering. It introduces a pattern language for Performance Evaluation, which is a combination of various analysis and design patterns modeled using a new modeling technique called "Software Stability" (Fayad, 2002). These patterns are mainly in the area of Performance Evaluation. Because the patterns obtained help to overcome several drawbacks seen in contemporary patterns, which will be discussed later, it implies that these can be used in several different applications. This thesis has four major contributions:

1. Analysis and design patterns are modeled using the concepts of "Enduring Business Themes" (EBTs) and "Business Objects" (BOs), in the field of Performance Evaluation. The analysis pattern, ***Evaluation,*** and the design pattern, ***AnyPerformance***, are provided in Appendix A.
2. A pattern language map for Performance Evaluation defines a unique model showing the classification, properties, development and deployment patterns.

3. Applicability of the obtained analysis and design patterns for different applications with implementations in various scenarios are provided in Appendix B.

4. An architectural pattern for Performance Evaluation, which is also contribution to the field of patterns, is provided in Appendix A.

All these are discussed in detail in the later chapters. In general, this thesis is expressive enough to understand the interrelationships between several analysis and design patterns for Performance Evaluation. Under certain conditions, it is the most reliable and stable model for all the patterns needed. Few published patterns are quoted in references section for this chapter (Fayad, Pradeep, 2003) and (Fayad, Pradeep, Sidiqqui, 2003).

1.5 Structure and Organization

Performance Evaluation is presented for various domains in the following chapters. These chapters are a comprehensive survey of new material.

Chapter 1, Introduction, presents a high-level description of the thesis "Pattern Language for Performance Evaluation." Chapter 2, Overview, provides comparisons between the pattern language and the existing methodology used for designing the patterns. A detailed account of how the developed patterns are an improvement over the contemporary patterns is debated in this section. Chapter 3, Classifications and Goals, discusses the goals/missions/classification/or partitions of the pattern language for performance evaluation. Chapter 4, Capabilities and properties, describes the pattern language map, along with the properties for the classification identified. Chapter 5, Performance Evaluation Knowledge Map, provides an interrelationship between patterns, how they interact, and their properties, along with the different routes that can be adopted for Performance Evaluation. Chapter 6, Development and Deployment, provides an overview of how the analysis and design patterns obtained can be implemented and deployed in different applications. Chapter 7, Conclusion, provides a conclusion for the thesis, Pattern Language for Performance Evaluation. Appendix A, Section A.1, ***Evaluation*** Analysis Pattern, describes the analysis pattern ***Evaluation***; Section A.2, ***AnyPerformance*** Design Pattern, describes the design pattern ***AnyPerformance***; Section A.3, Performance Evaluation Architectural Pattern, describes the stable architectural pattern for performance evaluation. Appendix B, Application Scenarios, provides the application scenarios with screen shots for implementing the analysis and design patterns, ***Evaluation*** and ***AnyPerformance***. Appendix C, Code Sample, provides the code samples for implementing the analysis and design patterns, ***Evaluation*** and ***AnyPerformance***.

CHAPTER 2

2.1 Overview of Pattern Language for Performance Evaluation

To overcome the issue of redesigning an entire system to handle new functionalities, it becomes important to develop a core system that usually exhibits a rich set of behaviors capable of handling features that are reusable, adaptable, and maintainable with minimal modifications. These result in recurring patterns of classes that solve specific design problems and make systems more flexible and ultimately, reusable. According to Christopher Alexander,

> "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to the problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" (Alexander, Silverston, Jacobson, Fiksdahl and Angel, 1997).

Sets of such related patterns comprise a patterns language, which is a system used for representing ideas and states related to patterns. In other terms, it is a set of certain cognitive processes involved in producing and understanding different concepts of patterns in various disciplines. Pattern language is usually characterized according to aspects and nature of different stable patterns. The common aspects that all stable patterns share are identified and as a result, each pattern category with its own aspects is obtained.

There are several methodologies available for developing patterns, (Alur, Crupi, and Malks, 2001), (Gamma, Helm, Johnson, and Vlissides, 1994), which systematically name, explain, and evaluate an important and recurring design in a software system. But sometimes, developing a good pattern with an appropriate methodology is expensive and may have drawbacks. Moreover, it may not necessarily be applicable to multiple domains. Stable Patterns Methodology, that handles the methodologies for developing both analysis and design patterns is used for developing patterns. The concept of Software Stability Modeling is used in these cases (Fayad, 2002). The patterns thus developed will be simple enough to learn and still provide enough features that can be used quickly to hook multiple applications with few changes. This also ensures that the pattern is flexible enough to be applied to many domains and remains stable even after addressing new requirements and design changes. Let us next examine why software stability patterns are an essential modeling technique.

2.2 Overview of Software Stability and Patterns

Software Stability modeling is a modeling technique that uses the concepts of "enduring business themes" (EBTs) and "business objects" (BOs). These are classes that are usually classified into three different layers. They are:

Layer 1: Enduring Business Themes (EBTs), which represents the core knowledge of the system

Layer 2: Business Objects (BOs), which form the concrete classes of the system and are generally called the "work horses" of the system under consideration

Layer 3: Industrial Objects (IOs), represent the external applications that can be hooked to the pattern and usually form the leaves, which can be replaced without affecting the system

Figure 2 gives an overview of different layers obtained for performance evaluation. The EBT layer contains ***Measurement*** and ***Evaluation***, which cover the core concepts. The BO that is shown here is ***Performance*** since our main focus is performance evaluation. There are several other BOs that are not listed here. Because the pattern that is developed uses a layered approach, it is important to understand these layers and how these layers are obtained.

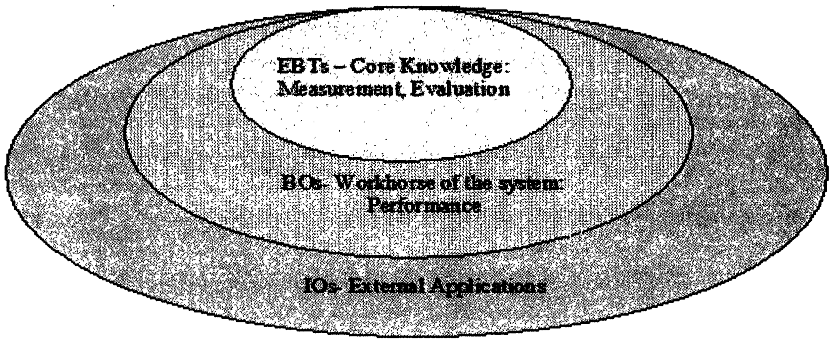


Figure 2: Software Stability Modeling Layers

To obtain EBTs of the system under consideration, those aspects of the system that remain stable over a period of time are identified. These are usually those aspects that form the core of the system based on what the system needs are and why the system is modeled. This helps in reducing the reengineering and code modifications required when requirements change. On the other hand, BOs are those classes that act as "workhorses" and form concrete classes. They are classes that change only internally but remain stable externally. As a result of obtaining these classes, 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 because 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. Hence, this reduces the changes that need to be made to the system when the requirements change.

Stable analysis and design patterns are conceptual models that are obtained as a result of capturing the core aspects of the problem under consideration. Patterns may vary in the level of abstraction and granularity. Capturing the core of the problem is mandatory to design the right solution. To successfully reuse the same model to address a similar problem, regardless of the nature in which the problem appears, it is important to extract features that are adaptable to handle multiple domains.

2.3 Organization of Pattern Language

The main objective of pattern language is to provide an exhaustive material of related patterns in the field of performance evaluation. Pattern language mainly consists of four different phases. These mainly include the classification or the core knowledge, properties, which also provide a description of the benefits, development, and last but not the least deployment. We consider the example Performance Evaluation in this paper to thoroughly understand these different phases or steps. Each of these phases is a set of patterns that uniquely identifies that phase. Figure 3 shows how these phases form the pattern language for performance evaluation.

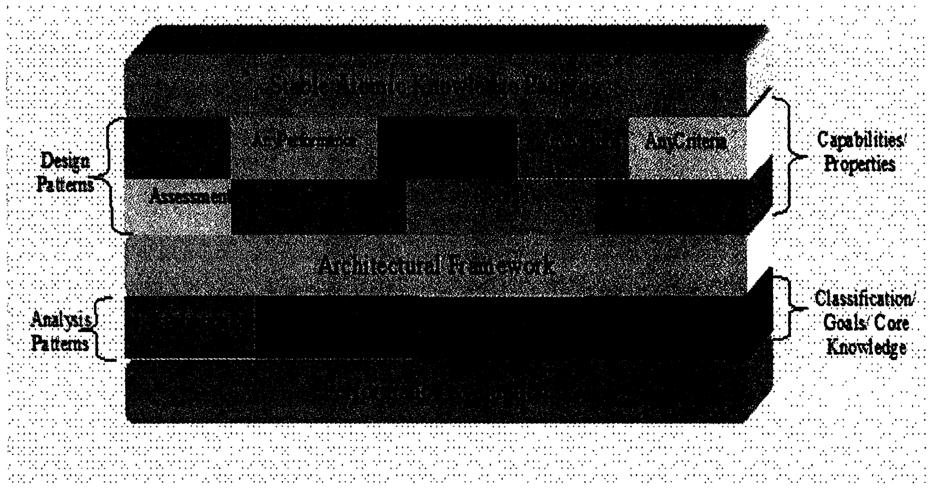


Figure 3: Performance Evaluation Categorization

As shown in Figure 2, Performance Evaluation forms the basis of this structure for which the relevant analysis and design patterns need to be obtained. The next layer gives all the EBTs, which form the analysis patterns and support the most common tasks in performance evaluation. These are identified as ***Evaluation***, ***Measurement***, ***Collection***, and ***Analysis***. The next layer, which sits on top of this layer, is the architectural framework. This layer indicates that the layers above and below this layer, which are the analysis and design patterns, collectively form the architectural framework for performance evaluation. The layer above this is the list of all BOs, which form the design patterns and are mainly identified as ***Assessment***, ***Metrics***, ***Storage***, and ***Optimization***. These uniquely identify the properties for EBTs obtained earlier. The other layer, which lies above this layer, provides a list of common patterns that help in achieving the goals specified by the EBTs in Performance Evaluation. The details on each layer are provided in the following chapters. Table 1 gives an overview of the various analysis and design patterns for Performance Evaluation.

Patterns	Class Type	Pattern Type	Related Design patterns
Evaluation	EBT	Analysis	AnyPerformance, Assessment
Measurement	EBT	Analysis	AnyPerformance, Metrics
Collection	EBT	Analysis	Storage
Analysis	EBT	Analysis	Optimization
Assessment	BO	Design	AnyParty, AnyEntity
Metrics	BO	Design	AnyPerformance
Storage	BO	Design	AnyMedia, AnyEntity
Optimization	BO	Design	AnyEntity, AnyParty

Table 1: Patterns for Performance Evaluation

As shown in the table, there are fours analysis patterns and four different design patterns obtained for Performance Evaluation. The map, which is provided in chapter 5, gives a more detailed picture of various analysis and design patterns and phases in pattern language.

2.3 Comparison with existing work

The main objective of this section is to compare and contrast this thesis work with existing material. There are several process models and techniques available for Performance Evaluation with respect to software performance engineering. For comparison, consider the discussion from Performance Evaluation and Monitoring (Henry, 1971), a paper that deals with techniques to evaluate performance. This discussion mainly handles the performance evaluation of a computer. Several techniques like benchmarks, analytical models, simulation, etc., have been proposed. But there is no pattern defined that can be used for performance evaluation of a computer in terms of speed, multiprogramming and multiprocessing features. Using software stability modeling approach to model the patterns for performance evaluation mainly provides a stable pattern that can be applied to any domain that uses Performance Evaluation. It also has the flexibility of providing hooks to these patterns that can handle any modifications. This can include discarding several existing functionalities or encompass new features. These patterns are also generic enough to be applied to any application. This might include providing links to the existing code and modifying it to hook to the new domain specific features. Therefore, using patterns modeled using Software Stability technique has several advantages over processes and techniques that already exist.

CHAPTER 3

3.1 Goals and Classification for Performance Evaluation PL

Classification, which also represents the core knowledge of the system, mainly contains those patterns that deal with cooperating EBTs (Fayad, 2002) that make up the pattern language (Fayad, Pradeep, 2004). A number of analysis patterns can be obtained that are related to performance evaluation. Phase 1, which deals with the activities involved in extracting the core knowledge of the system for Performance Evaluation is addressed in the next section. It illustrates the steps involved in obtaining the relevant EBTs for Performance Evaluation.

3.2 Phase 1: Identification of Goals, Classifications and Missions

Phase 1 is the first step that needs to be considered when the pattern language for performance evaluation is developed. It illustrates a step-by-step approach to finding the right stable analysis patterns (Hamza, 2002), (Hamza, Fayad, 2002a), (Hamza, Fayad, 2002b), (Hamza, Fayad, 2002c), required for Performance Evaluation.

- ***Consider what the system is intended to do:*** This helps to obtain those patterns that lead to the core knowledge and the objective of the system. For example, the system is required to evaluate some entity based on the collection of data obtained from a particular task and provide the results of evaluation as a measurement, based on some algorithms that are used for following a procedure, or optimizing the existing processes etc.

- ***Obtain relevant patterns:*** Based on the system's intention, obtain relevant patterns that will solve the problem under consideration.
- ***Obtain pattern interrelation:*** Once the patterns are obtained, it is important to study how these patterns are related to each other. Studying this relationship will help in obtaining the right group of patterns. For example, examine if the patterns obtained for performance evaluation are really related to each other in a way they are supposed to be.
- ***Examine a redesign cause:*** Consider and speculate the causes that might lead to redesigning of the system and the way the patterns relate to each other. This will help in obtaining a broader perspective of the problem and how the patterns that are already obtained can encompass these issues.

Because the goal of Performance Evaluation is to evaluate the performance of any task performed by an entity, ***Evaluation*** (Fayad, Pradeep, 2003) is one of the EBTs needed as it addresses the goal of evaluation. When a task that is to be evaluated is performed, the data or metrics obtained from it is handled by another EBT, ***Measurement***. Another related EBT for performance evaluation is ***Collection*** as the data collected is stored for further processing. This addresses the goal of collection of any kind of information or data and handles related collection activities. ***Analysis***, on the other hand, includes all the analysis techniques needed for performance evaluation process. These EBTs, together with the BOs, provide a generic framework that can be used to develop implementations based on the application by providing relevant hooks.

The related BOs, the workhorses of the system, are addressed in the next chapter. Table 1 provides a list of EBTs required for Performance Evaluation.

Goals/Missions/Classification/ Stable Analysis Patterns	Class Type	Description
Evaluation	EBT	Defines the process of judging the competence with which a party has performed the task assigned to it or the process of data collection and analysis to determine the success or failure of performers for a specific task, as a result of performance
Measurement	EBT	Represents the process of obtaining data for measuring any task or process performed by an entity
Collection	EBT	Represents the collection of any data that can be obtained during the process of performance evaluation. It includes all the relevant information that can be used for any analysis that needs to be done once a task is performed and evaluated.
Analysis	EBT	Represents all the analysis techniques used for performance evaluation process. This can include all the processes that perform analysis of the tasks that are evaluated or the analysis of the performance evaluation process itself.

Table 2: Classification patterns for Performance Evaluation

To provide concrete functionality for any system, the relationship between different EBTs needs to be identified based on the system needs. The EBT Evaluation interacts with Measurement to obtain the metrics details for a particular task performed. This might need to interact with Collection EBT to obtain the stored results. Analysis is required only when analysis of the evaluation report from ***Evaluation*** analysis pattern is needed.

3.3 Conclusion

The Classification and Goals category in developing pattern language for Performance Evaluation helps provide all the stable analysis patterns that help the system in evaluating performance. Phase 1 provides an overview of the steps and activities involved in identifying the goals of the system. The patterns obtained in this category form the stable analysis patterns.

CHAPTER 4

4.1 Capabilities and Properties for Performance Evaluation PL

The category capabilities and properties for Performance Evaluation pattern language represents all the properties exhibited by the patterns identified during classification. This category uses phase 2 for identifying the properties, represented in terms of patterns, for the goals specified in the Classification category. Phase 2 is important because it helps obtain the workhorses of the system under consideration. The next section deals with this phase in detail.

4.2 Phase 2: Identification of Properties and Capabilities

This phase helps in identifying the design patterns, which reflect the BOs of a system under consideration. Because these are to remain stable and still provide the functionalities of the workhorses, helping EBTs achieve the goals, keeping them general enough to address core issues is important. Hence, these are usually abstract classes with methods to achieve general functionality. These are extended to provide hooks, to suit the application needs. Certain steps are given below for finding the set of capabilities for each goal or a set of properties for each pattern obtained during classification.

- *Read each pattern to obtain an overview*: Each pattern, for which the properties have to be obtained, has to be understood thoroughly by paying particular attention to the sections - Consequences, Results, Tradeoffs, and Solutions in the pattern template.

- *Study the CRC cards, Participants, and Pattern structure*: Emphasize on the classes in the pattern to understand how they relate to each other with respect to EBTs and their BOs. This provides the possible patterns that might represent the properties of the pattern.
- *Consider applicability of the pattern*: Check to see which domains the pattern can be applied to in order to obtain the properties appropriate to encompass the domain features.
- *Identify the patterns in properties*: Once the properties are identified, examine the recurring classes in those properties to check if they form a pattern. Choose the right name for the pattern that will appropriately signify the property of the pattern under consideration.

In the process of performance evaluation, there are certain BOs that actually help EBTs accomplish their goals. These can be categorized as unique properties and general properties for the EBTs. ***AnyPerformance*** pattern (Fayad, Pradeep, Siddiqui, 2003) falls under the general property in this phase because it does not uniquely identify any EBT. But it is a type of measurement and is hence associated with ***Measurement***. Patterns ***Assessment*** and ***AnyMetrics*** are uniquely associated with ***Evaluation*** (Fayad, Pradeep, 2003) because they signify the function of evaluation. Similarly, ***Optimization*** identifies ***Analysis*** and ***Storage*** identifies ***Collection***. There are several other patterns like ***AnyParty*** (Fayad et, al. 2005), ***AnyEntity*** (Fayad et, al. 2005), and ***AnyConstraints***, which are used commonly used in all the analysis patterns. The details of their relationship and the overall patterns interaction are explained in detail in the next chapter, Pattern Language Knowledge Map. Here are several different patterns identified during classification with their properties.

Classification	Capabilities/Properties	Description
Evaluation	Assessment, Metrics	Represents the process or the property of observing and evaluating a party's performance, recording the assessment, providing the status, findings, evidence, and result of any evaluation process. It also indicates how assessment is influenced by the constraints or the external factors that are introduced in the system
Measurement	Metrics	Describes performance evaluation as a type of measurement, which provides the metrics or measurements of the task that is evaluated
Analysis	Optimization	Represents the attributes of all the optimization techniques that are used in any analysis to enhance its utilization and the process of performance evaluation as a whole
Collection	Storage	Represents the property of storage used by data collection for the purpose of performance evaluation

Table 3: Patterns for Capabilities for Performance Evaluation Pattern Language

4.3 Conclusion

Properties and capabilities play an important role in defining pattern language for Performance Evaluation. Phase 2 in developing pattern language helps in deriving the workhorses of the system. These workhorses, termed Business Objects, are usually represented by stable design patterns.

Chapter 5

5.1 Performance Evaluation Pattern Language Knowledge Map

The pattern language map (Hamza, Fayad, 2002) provides the interrelationship between patterns, how they interact, and their properties, along with the different routes that can be adopted to obtain architectural patterns based on the application needs.

Figure 4 shows the pattern language map for performance evaluation (Fayad, Pradeep, 2004). The main categories like Goals, Capabilities, Development and Deployment, for the pattern language map are given in rectangles within rectangles. The other rectangles represent the various aspects and patterns needed for Performance Evaluation. The categories Goals and Capabilities are explained in detail in Chapter 3 and Chapter 4 respectively. Development and Deployment are dealt with in the Chapter that follows next. The arrows in the map indicate the routes that can be taken to obtain sensible architectural patterns for any domain that requires these patterns. For instance, ***AnyParty*** (Fayad al, al. 2005) initiates the process of performance evaluation that needs to be done for ***AnyEntity*** (Fayad al, al. 2005), taking into consideration ***AnyCriteria*** that is specified by ***AnyParty***. The metrics obtained based on performance of ***AnyEntity*** are then sent to ***AnyMetrics***, which in turn is used by ***Evaluation*** for evaluation purposes. The figure for pattern language map for Performance Evaluation follows next.

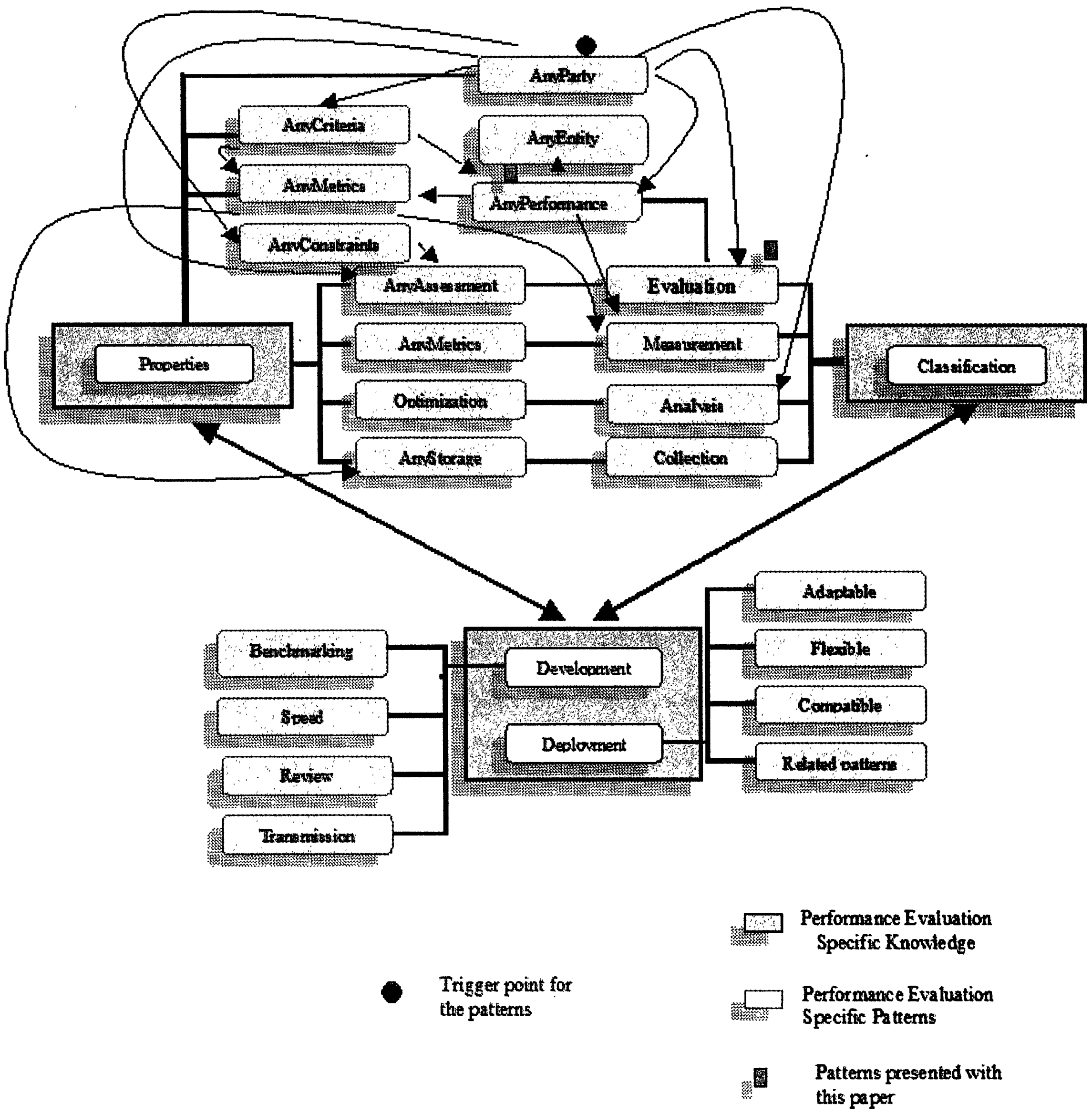


Figure 4: Pattern Language Map for Performance Evaluation

Section 5.2 provides a simple architectural pattern for performance evaluation

5.2 Architectural Pattern for Performance Evaluation

An architectural pattern is a result of using a particular route in the pattern language map. Thus, when a route, shown in the figure for pattern language map is adopted, an architectural pattern is for Performance Evaluation is obtained. The Figure 5 shows the interaction between the EBTs and BOs. This pattern contains two EBTs, ***Measurement*** and ***Evaluation*** and several BOs, ***AnyPerformance***, ***AnyCriteria***, ***AnyMetrics***, ***AnyEntity***, ***AnyParty***, and ***AnyAssessment***.

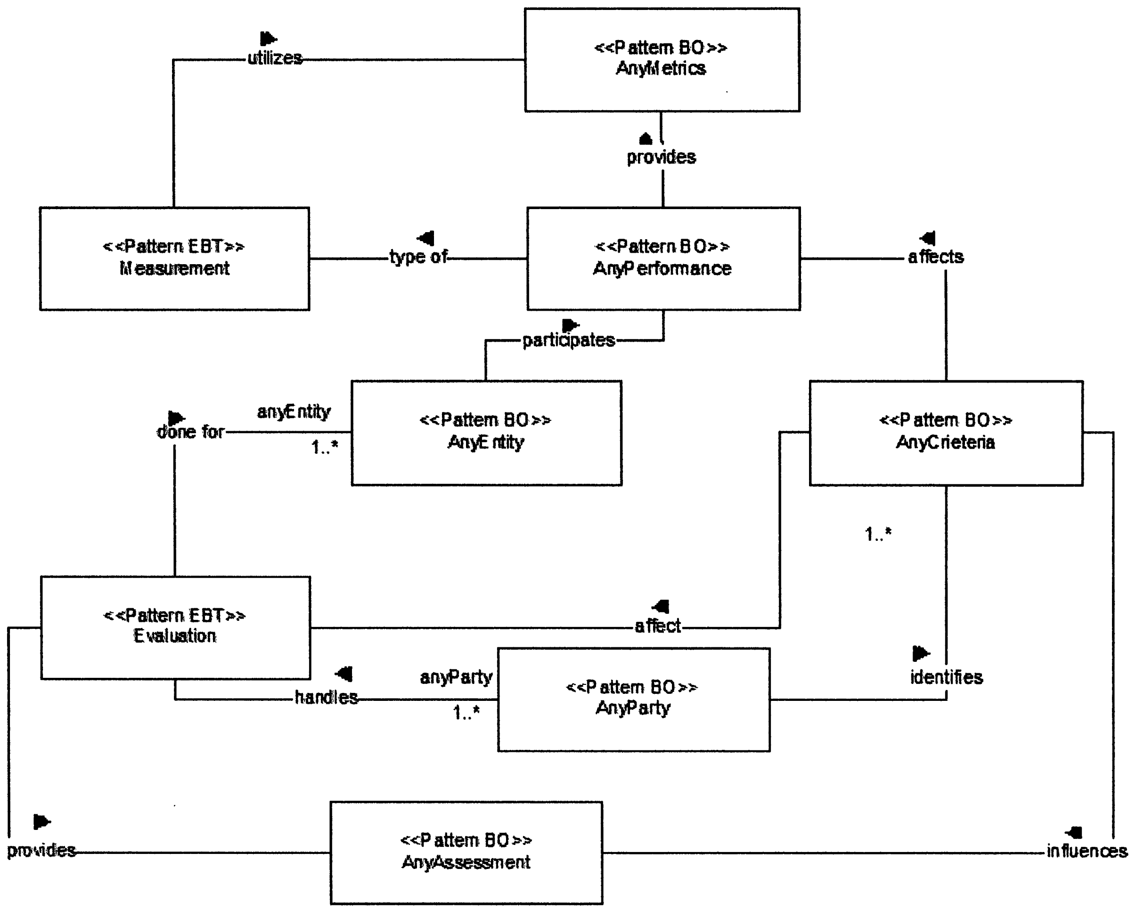


Figure 5: Architectural Pattern for Performance Evaluation

Each EBT forms an analysis pattern and the BOs, the design patterns. This pattern contains two EBTs, ***Measurement*** and ***Evaluation*** (Fayad, Pradeep, 2003), and BOs, ***AnyPerformance*** (Fayad, Pradeep, Siddiqui, 2003), ***AnyCriteria***, ***AnyMetrics***,

AnyEntity, ***AnyParty***, and ***AnyAssessment***. ***Evaluation*** and ***AnyPerformance*** patterns are provided in Appendix A.

The architectural pattern, ***Performance Evaluation***, deals with providing feedback for a activity that is time-bound and provides assessment based on the success or failure of that activity. This begins with ***AnyParty*** evaluating performance of ***AnyEntity***. This is based on ***AnyCriteria,*** the requirements or standards that ***AnyEntity*** needs to satisfy. ***AnyPerformance*** measures the performance of any task performed by ***AnyEntity*** and forwards the result to ***AnyMetrics***. The performance data from ***AnyEntity,*** stored in ***AnyMetrics***, is then used by ***Measurement***. ***Evaluation*** pattern provides assessment, ***AnyAssessment***, based on the criteria.

5.3 Conclusion

The pattern language map for Performance Evaluation provides all the relevant patterns required for performance evaluation. Any route can be chosen to from the set of patterns in the map to provide a meaningful architectural pattern. This chapter provides an example of an architectural pattern ***Performance Evaluation***.

CHAPTER 6

6.1 Development for Performance Evaluation Pattern language

The development category deals with the process of improving and developing the patterns obtained in capabilities and properties phases. The focus is to evolve or develop the patterns from the Goals and the Capabilities categories to present a more productive and meaningful stage. This phase provides a set of possible scenarios that can be obtained when different routes in the map are adopted. The routes result in different architectural patterns that are usually a combination of several EBTs and BOs (Fayad, 2002).

Consider the case of the architectural pattern ***Performance Evaluation*** (Fayad, Pradeep, 2004), to understand the concept of route better. The route, which this architectural pattern adopts, is shown in detail in the pattern language knowledge map. The EBTs ***Evaluation*** and ***Measurement*** are involved along with the BOs, ***AnyParty***, ***AnyEntity***, ***AnyPerformance***, ***AnyMetrics***, ***AnyAssessment***, and ***AnyCriteria***. The architectural pattern is involved in the complete cycle of evaluating the performance of any task performed. A few examples to illustrate the development route of architectural pattern for Performance Evaluation are shown in Table 4.

Development Route	Description	Application
Benchmarking	Represents a structured approach for identifying the best practices, comparing and adapting them to achieve intended results, and suggesting ambitious goals improvement.	Used in performance benchmarking in industries, government, devices performance, etc.

Speed	Describes the rate at which any specified operation is performed.	Used in computational speed of devices like computers, cell phones, loading and storing operations for memory, etc.
Review	Represents a formal or official examination for evaluation to polish performance.	Adopted for performance reviews in organizations, art, and technology
Transmission	Describes the process of transmitting information in any form from source to destination using a specified media.	Used for data and voice transmission in networks, etc.

Table 4: Development routes for performance evaluation knowledge map

6.2 Deployment for Performance Evaluation Pattern language

The deployment phase represents all the issues related to deployment. There are certain patterns that are used for representing this. They are discussed in the following section.

- ***Scalable:*** This pattern represents the properties related to scalability with respect to various patterns and the participants for the same.
- ***Adaptable:*** This presents the properties related to the adaptability of the developed patterns when they are deployed.

Only a few patterns related to deployment are mentioned here. The idea behind having different categories and phases is to provide the flexibility to choose the desired route in order to meet the requirements specified for an application. The next chapter concludes the thesis based on why software stability modeling (SSM) (Fayad, 2002) is an important methodology for designing patterns and systems on the whole.

6.3 Conclusion

Development and Deployment categories are an important part of the pattern language map since they give the user the flexibility to choose any route based on the application or system requirements. Architectural patterns are a result of choosing several meaningful, related EBTs and BOs.

REFERENCES

Alexander, C., Ishikawa, S., Silverston, M., Jacobson, M., Fiksdahl-King, I., & Angel, S., (1997). *A Pattern Language.* Oxford University Press, NewYork.

Deepak Alur, John Crupi, & Dan Malks (2001). *Core J2EE Patterns, Best Practices and Design Strategies*, Sun Microsystems Press, Prentice Hall.

Erich Gamma, Richard Helm, Ralph Johnson, & John Vlissides (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.

H. Hamza & M.E. Fayad (2002). Towards a Pattern Language for developing Stable Patterns. *Tenth Conference on Pattern Language of Programs (PLoP 03),* Illinois, USA, September 2002.

Henry C Lucus, Jr,. (1971). Performance Evaluation and Monitoring. *Computing Surveys*, Vol.3, No 3.

M. E. Fayad. (2002). Accomplishing Software Stability. *Communications of the ACM,* Vol.45, No 1.

M. E. Fayad & Rohini Pradeep (2003). Evaluation Analysis Pattern. *Stable Patterns, workshop # 8*, UML 2003.

M. E. Fayad & Rohini Pradeep. (2004). Pattern Language for Performance Evaluation. *SugarLoaf PLoP'04, The Fourth Latin American Conference on Pattern Languages of Programming*, Brazil, August 2004.

M.E. Fayad, Rohini Pradeep, & F. Siddiqui. (2003). Aspects in Communication: Performance. *Aspect-Oriented Modeling with UML, workshop #4*, UML 2003.

APPENDIX A: STABLE PATTERNS

A.1 Abstract

This section deals with how Stable Analysis Patterns (Hamza, Fayad 2002, 2003) can be used to develop patterns that can be used in Performance Evaluation (Fayad, Pradeep 2004). The obtained ***Evaluation, AnyPerformance, and Performance Evaluation*** patterns can be applied to any domain as a time-bound activity that attempts to assess systematically and objectively the relevance, performance, and success of an ongoing or completed activity. Because performance evaluation covers multiple domains, it is not easy to come up with a model that encompasses all the features. There are several methodologies available for developing patterns. Unfortunately, developing a good pattern with an appropriate methodology is expensive and has drawbacks, and may not necessarily be applicable to multiple domains. Stable Patterns provide a solution for developing patterns that will be simple enough to learn and still provide enough features that can be used quickly to hook multiple applications with minimal changes. This also ensures that the pattern is flexible enough to be applied to many domains and remain stable even after addressing new requirements and design changes.

A.2 Introduction

Patterns form an important part of any object-oriented analysis and design methodology. A pattern should be general enough to address and encompass future requirements and problems. Because a pattern needs to be reusable, avoiding redesign of the entire system, it becomes important to develop a good pattern that can be applied to multiple applications and domains. There are a number of methodologies available for developing patterns. But developing a good pattern with an appropriate methodology can sometimes be expensive with drawbacks. The developed pattern may not necessarily be suitable to be applied to multiple domains. This Appendix section deals with how Software Stability Modeling (Fayad, Altman, 2001), (Fayad, 2002), and the concept of Stability Analysis Patterns (Hamza, Fayad, 2002), provide a solution for developing a good pattern. The pattern thus modeled will be complete with enough features to provide flexibility and extensibility with hooks to attach to several applications. The following section provides the problem and the solution, along with the applicability for the pattern indicated earlier.

A.3 Evaluation Analysis Pattern

Evaluation Analysis pattern can be applied to any domain as a time-bound activity that attempts to assess systematically and objectively the relevance, performance and success of an ongoing or completed activity. Several functionalities can be evaluated using this pattern. Those functionalities can include: checks for accuracy, adaptability, changeability, clarity, compliancy, conformity, efficiency, effectiveness, feasibility, interoperability, maturity, operability, readability, recoverability, security, stability, suitability, or testability.

A.3.1 Problem

The ***Evaluation*** analysis pattern covers many domains that are different in nature and functionality. Therefore, modeling a generic analysis pattern that can be applied to these domains is a problem. Moreover, the requirements of each domain vary based on the users of the system and the participants. The entity that needs to be evaluated can be from different domains. For e.g., different generations of wireless networks need to be evaluated based on different features, lands need to be evaluated to verify their suitability for agriculture, etc. Hence, obtaining a model or a pattern that covers all the functionalities is a complex and tedious task. The main challenge that needs to be addressed is obtaining a model that handles these variations.

A.3.2 Context

Evaluation is an important concept in any domain that requires a response or feedback for a time-bound exercise that assesses various factors like relevance, performance, success or failure of an ongoing or completed activity in a systematic and objective way. The entity involved in the activity can be anything from an artist to an employee, an engine, or a network whose evaluation needs to be done by a party, essentially an evaluator like a person or a system. For example, an engine can be evaluated in an assembly line; the audience in a theatre can evaluate an artist. Wireless networks or networks in general can be evaluated for comparison based on certain constraints that influence the assessment provided. These constraints can be any factor that affects the overall judgment or assessment either directly or indirectly.

In general, ***Evaluation*** has several distinguishing characteristics that relate to focus, methodology, and function. Thus, ***Evaluation*** as a pattern, assesses the effectiveness of an ongoing activity in achieving its objectives, relies on the standards used by the activity to distinguish any other activity's effects or external constraints, and aims at the activity improvement through modification of current operations. This concept can be applied to multiple fields as already mentioned earlier.

A.3.3 Forces

Many contexts and domains that are completely different in nature are covered by the evaluation analysis pattern. Evaluation for any entity can be done by one or more entities simultaneously, based on multiple constraints. Hence, the pattern needs to handle multiple constraints and entities. The main challenge lies in handling different kinds of constraints and entities for different domains. Flexibility with respect to handling domain-specific features is also another challenge for this model. Assessments can be different based on the context it occurs in. The features for assessment are domain specific and addressing these features is a limitation faced by this model.

A.3.4 Solution

The stable pattern that is obtained in Figure 1 is a proposed solution for handling the issues mentioned earlier in the Problem [A.1.4] section. This concentrates on providing a generic pattern that can be applied to any domain, leaving out the domain specific features.

A.3.5 Pattern Structure

The relationship between (EBT), which is evaluation in this case and the (BOs), for evaluation analysis pattern is shown in the figure below.

Participants

The participants in the pattern structure are described based on the classes and patterns occurring within the evaluation pattern.

Classes

- ***Evaluation***: This is the core of the stable analysis model and represents a time-bound activity that attempts to assess systematically and objectively the relevance, performance, and success of an ongoing or completed activity process.

Patterns

- ***AnyParty***: This represents parties that are involved in the task of evaluation of any activity performed by any entity or in some cases, the entity itself and provide assessments for the same. This party can be an evaluator of some task that is already completed, or in the process of completion. The evaluator can be an employer, an intelligent process that evaluates any task, or a network agent evaluating the performance of the network elements.

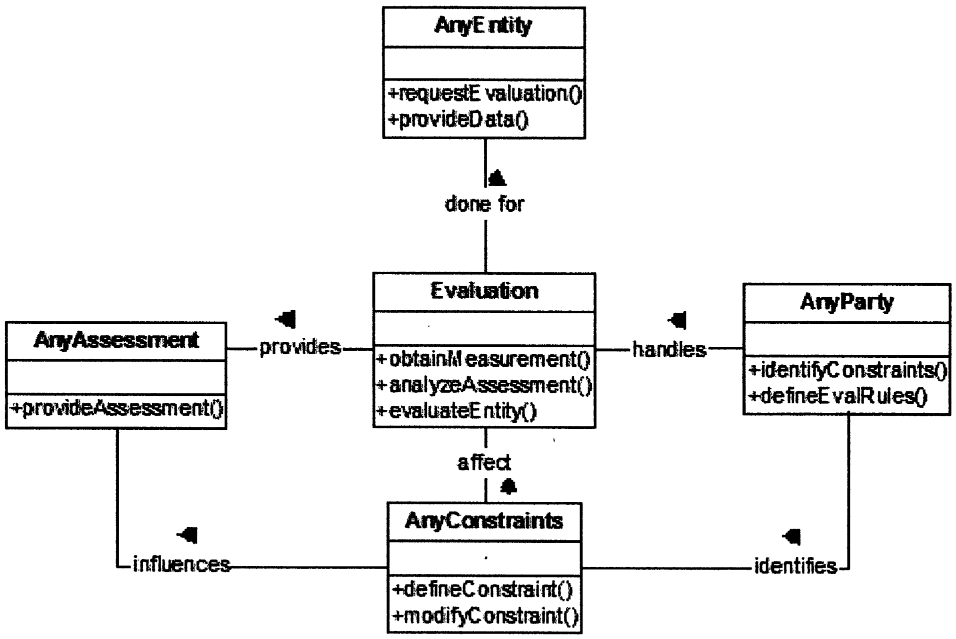


Figure 6: Evaluation Analysis Pattern

- ***AnyEntity***: This represents any entity that needs to be evaluated. It can be a wireless system, an engine part, etc., that needs to be evaluated. These are usually termed as the Evaluation requestor.

- ***AnyAssessment***: Provides the status, findings, evidence, and result of any evaluation process. It also indicates how assessment is influenced by the constraints or the external factors that are introduced in the system.
- ***AnyConstraints***: This provides the details of issues related to the external factors and the constraints that affect the effectiveness of an ongoing activity in achieving its objectives. It can also be the standard used by the activity to distinguish any other activity's effects or external constraints that need to be considered during evaluation.

A.3.6 CRC Cards

The CRC cards shown in the following tables provide details on collaboration and responsibilities of various participants in the pattern.

Evaluation (Evaluation facilitator)		
Responsibility	Collaboration	
	Client	Services
Represents a time-bound activity that attempts to assess systematically and objectively the relevance, performance, and success of an ongoing or completed activity process	AnyParty, AnyEntity, AnyAssessment, AnyConstraint	defineEvaluation(), analyzeAssessment(), evaluateEntity( )

Table 5: CRC Card for Evaluation

AnyEntity (Evaluation Requestor)		
Responsibility	Collaboration	
Requests evaluation to be done based on the criteria and standard set	Client	Services
	Evaluation	requestEvaluation(), provideData()

Table 6: CRC Card for AnyEntity

AnyParty (Evaluator)		
Responsibility	Collaboration	
	Client	Services

Setting the rules along with

identifying the constraints under which the evaluation of any entity is done	Evaluation, AnyConstraints	identifyConstraint(), defineEvalRules()

Table 7: CRC Card for AnyParty

AnyConstraints (Constraint Definer)		
Responsibility	Collaboration	
	Client	Server
Describes the rules, standards and the external factors that need to be considered before any task can be performed	Evaluation, AnyParty	defineConstraint() modifyConstraint()

Table 8: CRC Card for AnyConstraints

AnyAssessment(Assessment Provider)		
Responsibility	Collaboration	
	Client	Server
Provides the status, findings, evidence, and result of any evaluation process	Evaluation, AnyEntity, AnyConstraints	provideAssessment ( )

Table 9: CRC Card for AnyAssessment

A.3.7 Consequences

The pattern has certain consequences that aid the pattern in supporting its objectives. They are listed below:

- Reduces Complexity: The evaluation pattern modularizes the various aspects involved in the evaluation process (example: assessment, constraints), and makes the pattern simple to understand.
- Facilitates adding new Industrial Objects: The pattern enables adding new "Industrial Objects" (IOs) based on the application domain. By enabling this capability, it provides the users of the pattern a core model that remains stable and encompasses all the important features of evaluation required in any domain.

- Provides Flexibility and Scalability: Applicable to various domains with sufficient flexibility and scalability in terms of constraints, entities, and parties.
- Presents Uniform Interface: The pattern provides a uniform interface to all applications that use evaluation. The underlying interactions between the EBTs and the BOs, which can be complex, are hidden, and a simpler interface that is easier to understand is presented instead.
- Improves adaptability and extensibility: The pattern is adaptable and provides a high level of extensibility to handle addition of new complex features.

A.3.8 Tradeoffs

The evaluation pattern has the following tradeoffs:

- Constraints Identification: The identification of constraints that influence the assessment process in evaluation is difficult to identify in certain domains. For example, when evaluating an artist in the theatre, there can be certain assessments provided on the basis of the artist's ethnic background or origin, which can be very difficult to capture in an actual code for constraints. The pattern has the drawback of its inability to take such constraints into consideration.
- 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.

A.3.9 Results

The modeling of the stable evaluation pattern results in a model with several advantages that resolve several issues related to ***Evaluation*** in terms of

- Generality: Obtained a generic pattern for evaluation that is applicable across various domains.
- Pattern Applicability: Aids for improving the existing evaluation process. This is done by providing modular layers in form of EBTs (evaluation) and BOs that form the core of any system that uses evaluation. The various aspects like assessment, and constrains related to the evaluation process are all handled in a single pattern.
- Reusability: Promotes a cleaner partitioning of EBTs like evaluation, which remains stable throughout and BOs that can change dynamically internally, thus encouraging reuse. The Industrial Objects for various applications can be added and removed transparently from the existing model without affecting the standard interface. Thus, any combination of industrial objects related to assessment, party, entity, and constraints can be used with the pattern with few modifications.

A.3.10 Applicability with illustrated examples

Problem Description

The pattern can be used for evaluation of the task of transmission of data in networks, speed of computer processing, order completion status, etc.

Problem Class Diagram

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

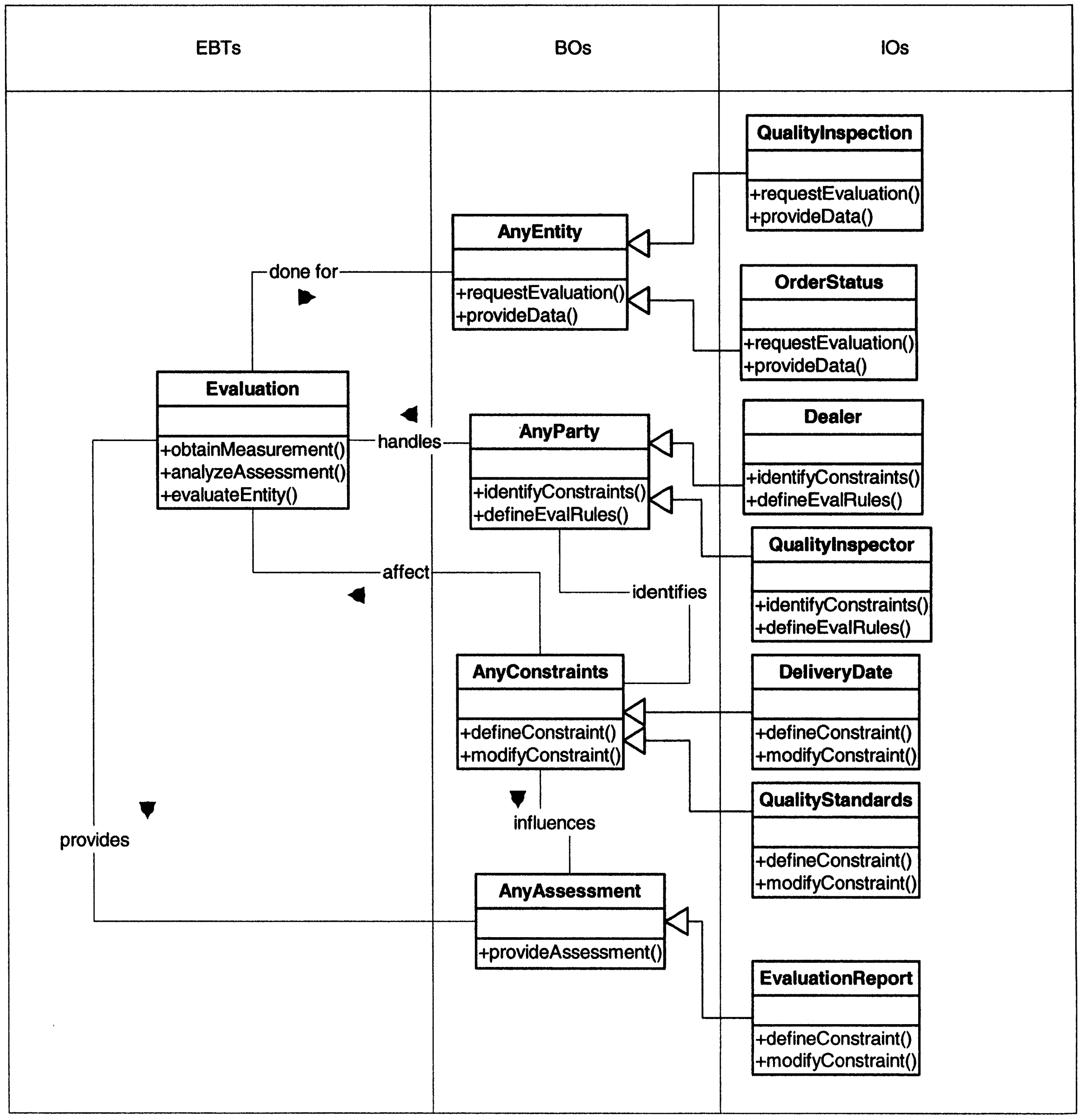


Figure 7: Evaluation Class Diagram for Manufacturing Industry

The next figure shows the class diagram for evaluation pattern when used for checking the performance of running a typical game application. AnyParty in this case is ScenarioBenchmark, which provides evaluation based on the AnyConstraints, ThresholdValues in this case.

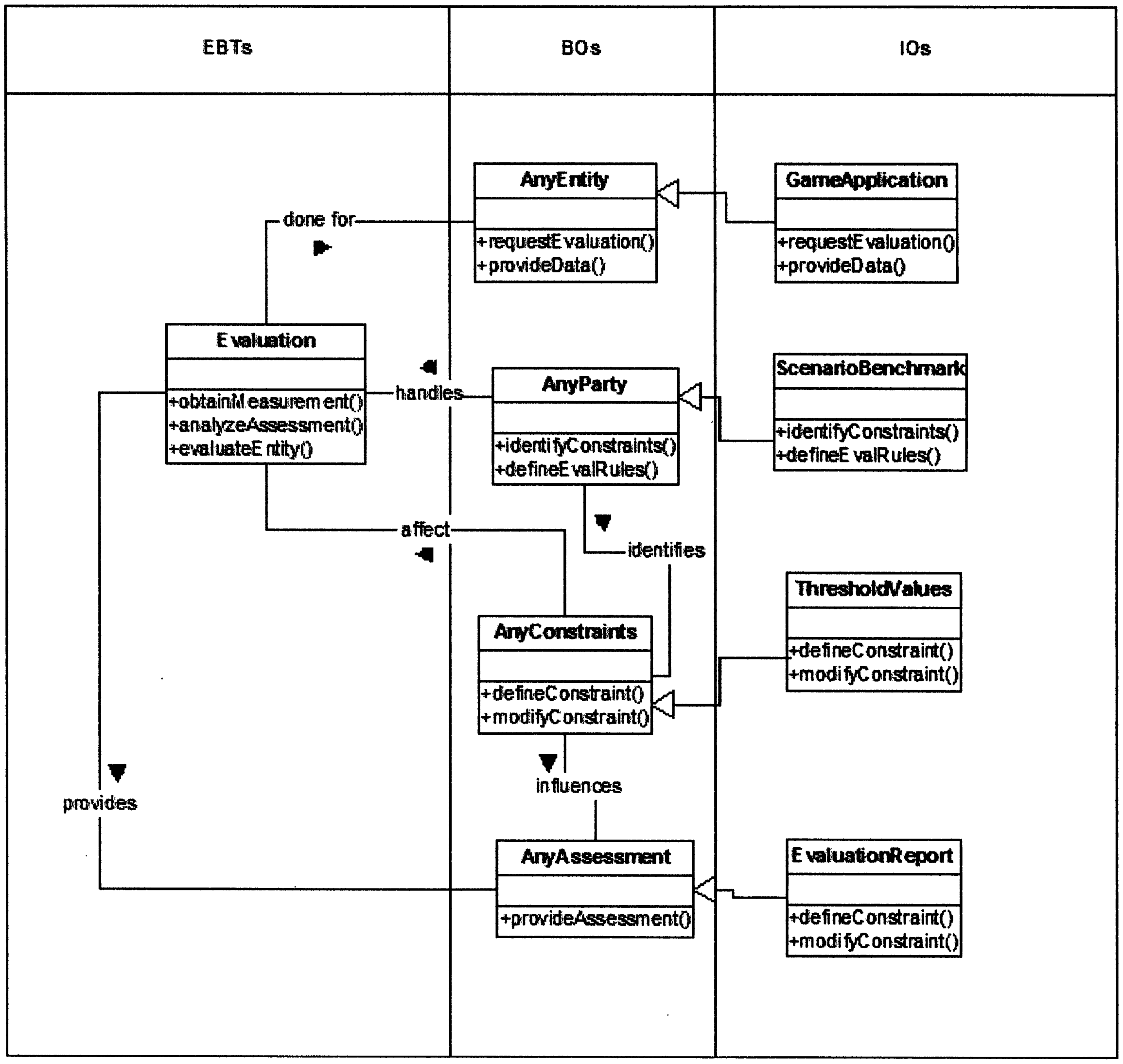


Figure 8: Evaluation class diagram for running a Game Application

A.3.11 Related Patterns

There are several patterns that usually interact with the above described analysis pattern. They are usually classified as:

Related Analysis Patterns

- ***Measurement***: Represents the process of obtaining data from measuring any task or process performed by an entity.
- ***Collection***: This represents the collection of any data that can be obtained during the process of evaluation. It includes all the relevant information that can be used for any analysis that needs to be done once a task is performed and evaluated.
- ***Analysis***: The pattern includes all the analysis techniques used for evaluation. This can include all the processes that perform analysis of the tasks that are evaluated.

A.3.12 Design/ Implementation Issues

Hooks Description

This section provides the description on various hooks that can be attached to this pattern.

- ***AnyParty***: The party that identifies an evaluator can be extended, by attaching an IO. For example, the party or an evaluator for a wireless network can be an operator who can change over a period of time and get replaced by an administrator. For the land evaluation process, the evaluator can be a different party in charge of carrying out the evaluation task. Thus, the party can be in any domain without requiring the entire code to be rewritten for a particular domain.

- ***AnyAssessment***: This BO can have a hook attached to it to encompass different assessments that can be associated with that domain. For example, in a network scenario, the assessments can change, while the pattern remains stable. That is, the Industrial Objects (IOs) for assessment change with minimal modifications required to encompass these changes.
- ***AnyConstraints***: This can have hooks attached to it to handle external factors and the constraints that affect the evaluation process. These can be considered IOs that can be replaced or modified without affecting the structure of the overall system. Since it can also represent standards that can be modified or updated based on the requirements of a domain, hooks can be provided to handle these changes.

A.4 AnyPerformance Design Pattern

A.4.1 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. Because it begins with "Any", it indicates that any domain that has any kind of performance involved can use this.

A.4.2 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.

A.4.3 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 a 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 can be a criterion objective or an enabling objective. For example, the domain based on the type of the services desired may specify the objective for a network in performing a particular task. 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.

A.4.4 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 ***Anyperformance*** pattern with a few changes. The Figure 6 shows the diagram of the ***AnyPerformance*** pattern.

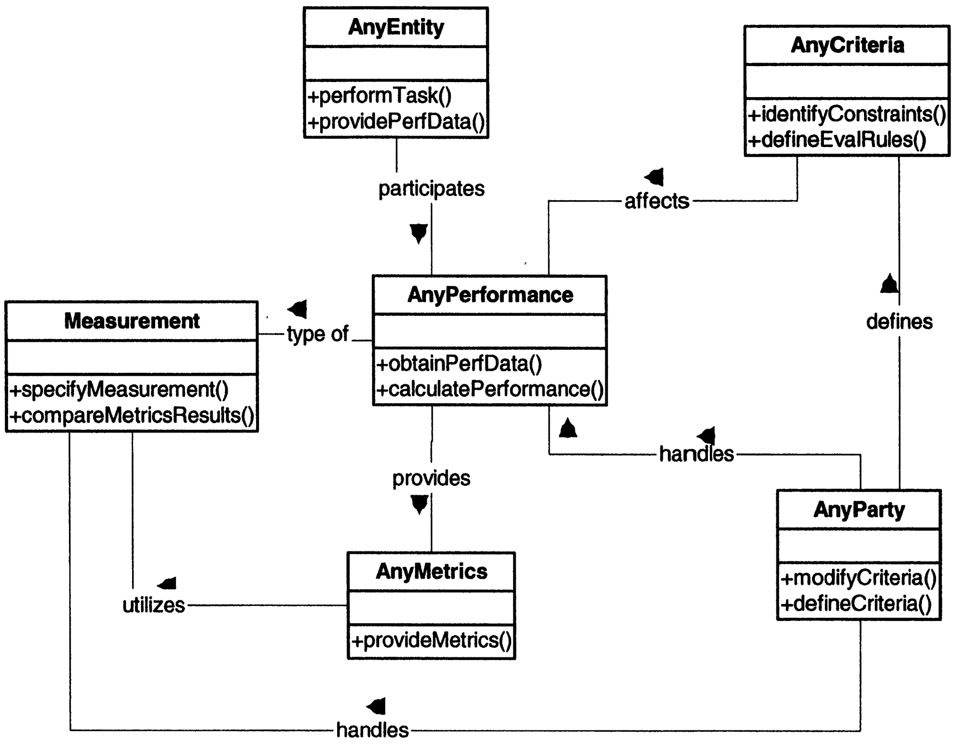


Figure 9: AnyPerformance Design Pattern

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

Classes:

- ***Measurement***: This represents the EBT that the BO performance is derived from. 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.

Patterns:

- ***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.

- ***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 observing 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. 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 or an engine that needs to finish or perform a task assigned. 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 directly or indirectly. It can be a standard that needs to be achieved, some conditions, and a reason for the performance that needs 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.

A.4.5 CRC Cards

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

Table 10: CRC Card for AnyPerformance

AnyEntity (Performer)		
Responsibility	Collaboration	
	Client	Services
Performs a task to a standard set.	AnyPerformance	performTask(), providePerformanceData()

Table 11: CRC Card for AnyEntity

AnyParty (Performance Requestor)		
Responsibility	Collaboration	
	Client	Services
Setting the rules under which the performance of any entity is measured.	AnyPerformance, AnyCriteria	evaluateEntityPerformance ( ), defineCriteria( )

Table 12: CRC Card for AnyParty

AnyCriteria (Criteria Definer)		
Responsibility	Collaboration	
Describes the rules, standards and	Client	Server

the external factors that need to be considered before any task can be performed.	AnyPerformance, Measurement, AnyParty	defineCriteria(), modifyCrietra()

Table 13: CRC Card for AnyCriteria

Measurement (Measurement)		
Responsibility	Collaboration	
	Client	Server
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.	AnyPerformance, AnyMetrics, AnyCritera	specifyMeasurement(), compareMetricsResults()

Table 14: CRC Card for Measurement

AnyMetrics (Metric Provider)		
Responsibility	Collaboration	
	Client	Server
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.	AnyPerformance, AnyEntity, Measurement	provideMetrics ( )

Table 15: CRC Card for AnyMetrics

A.4.6 Consequences

Pattern supports its objectives in the following ways

- The pattern achieves its objectives by providing a generalized process for performance evaluation in various domains with sufficient flexibility.
- This is applicable to various domains that use performance.
- Sufficient flexibility and scalability is provided by this pattern for performance.

The performance pattern has the following benefits:

- It is adaptable for different kinds 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 evaluate all the protocols developed from these standards.
- It is easy to use. The performance pattern developed here is a general pattern to be adapted for different parties and for different kinds of entities. This provides a high level of extensibility to handle adding new and complex features to this model.

A.4.7 Tradeoffs

The pattern ***AnyPerformance*** has the following tradeoffs:

- This pattern introduces a trade-off between generalization and flexibility.
- 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.

A.4.8 Results

- 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 the media.

A.4.9 Applicability with illustrated Examples

Problem Description

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

Problem Description

The pattern can be used for performance of the task of transmission of data in networks along with several other applications like industries, medicine, etc.

Problem Class Diagram

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.

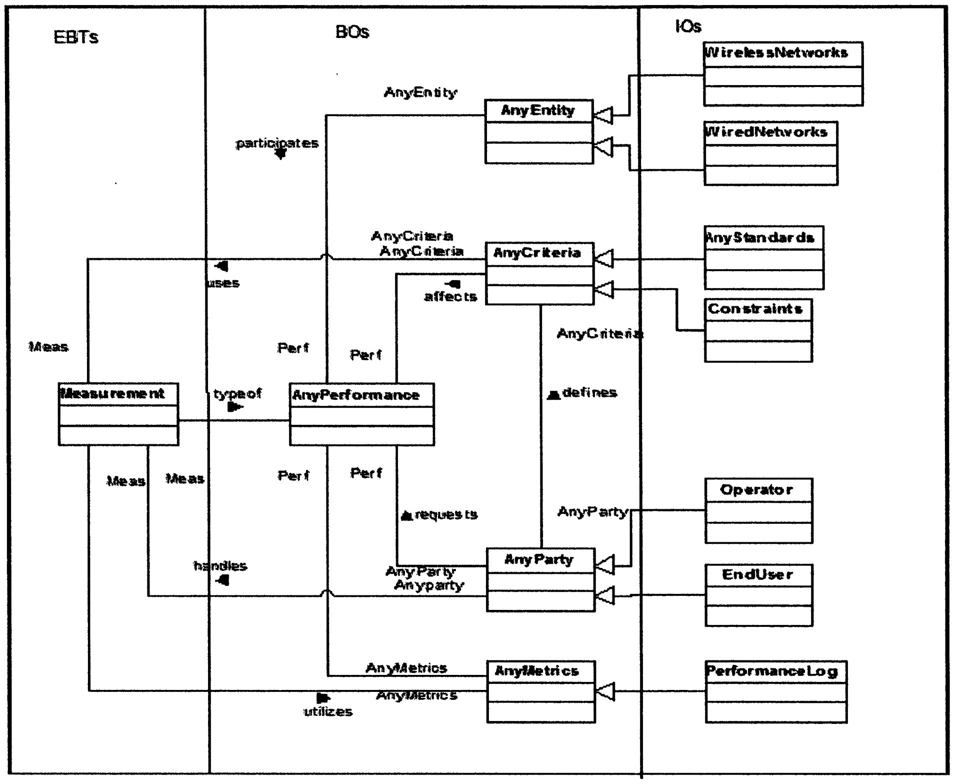


Figure 10: Performance Class Diagram for a Wireless Network

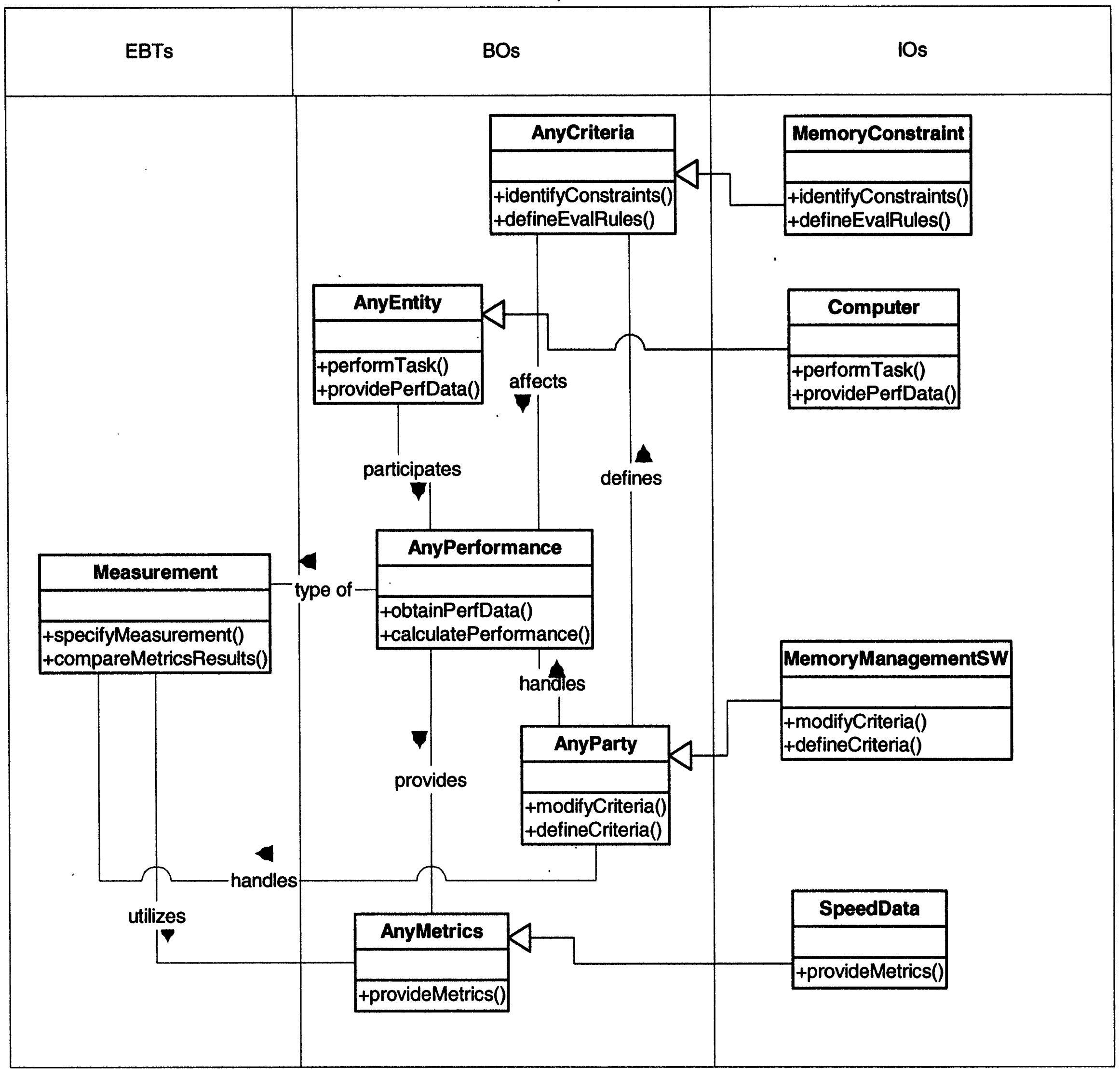


Figure 11: Performance Class Diagram for a Computer

A.5 Performance Evaluation Architectural Pattern

A.5.1 Performance Evaluation Architectural Pattern

Performance Evaluation architectural pattern can be used with any application that involves evaluating the performance to assess the relevance, along with the success or related results of an ongoing activity.

A.5.2 Problem

The ***Performance Evaluation*** architectural pattern covers domains that are different in several respects, which include functionality and other aspects. Hence, obtaining a pattern that encompasses and addresses these functionalities is a problem. Moreover, the requirements of each domain vary based on the users of the system.

A.5.3 Context

The aspect ***Performance Evaluation*** is of vital importance in any area that requires selection of an entity or design and development of new applications based on the performance of existing ones. Evaluation in these cases is done with a specific goal in mind, i.e., to assess the performance of an activity. In some case, the goal of evaluation can be to maximize the throughput of any activity at a reduced cost. For example, the throughput of a computer system may be evaluated or workload of the same system can be maximized at a lower cost. Thus, several special features can also be evaluated based on certain aspects that affect the considered system either directly or indirectly.

In general, ***Performance Evaluation*** has several distinguishing characteristics that relate to focus, methodology, and function. And ***Performance Evaluation*** as a pattern assesses the effectiveness of an ongoing activity in achieving its objectives, while relying on the constraints specified.

A.5.4 Forces

Because many applications that are different in nature are covered by ***Performance Evaluation*** architectural pattern, the main challenge lies in thoroughly understanding the requirements and aspects involved. Evaluation is usually undertaken with a specific goal and purpose, and the techniques employed must be considered in light of the objectives of the evaluator. Obtaining these objectives and handling them is a difficult task to accomplish. Assessments can be different based on the context it occurs in. The features for assessment are domain specific and addressing these features is a limitation faced by this model.

A.5.5 Solution

The stable architectural pattern that is obtained in the figure below is a proposed solution for handling the problems mentioned earlier. This concentrates on providing a pattern that can be applied to any domain, leaving out the domain specific features.

A.5.6 Pattern Structure

The relationship between different layers of stability model, which are EBTs like ***Evaluation*** and ***Measurement***, along with the BOs like ***AnyPerformance***, ***AnyAssessment***, ***AnyMetrics***, ***AnyParty*** and ***AnyEntity*** is shown in the figures below.

Participants

The participants in the pattern structure are described based on the classes and patterns occurring within the evaluation pattern.

Classes

- ***Evaluation***: This represents the process of evaluation of a time-bound activity that attempts to assess the performance of an ongoing activity in an objective way.
- ***Measurement***: This represents the EBT that the business object performance is derived from. It utilizes the data and the observations that are obtained as a result of a task

Patterns

- ***AnyParty***: This represents parties that are involved in the performance evaluation process. They can either request evaluation of an activity or evaluate an activity that is already completed or in the process of completion. For example, the party can be an employer evaluating the performance of an employee or a scenario benchmark that is requesting evaluation of a typical game in a wireless device.
- ***AnyEntity***: This represents any entity whose performance needs to be evaluated. It can be a wireless system, an engine part, or a computer that needs to be evaluated.
- ***AnyAssessment***: Provides the results and findings for performance evaluation, along with relevant status. It also gives an indication of the effect of the external factors that are introduced in the system.

- ***AnyCriteria***: This provides the details of the external factors, like the standards and constraints that affect the effectiveness of an ongoing activity in achieving its objectives.
- ***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

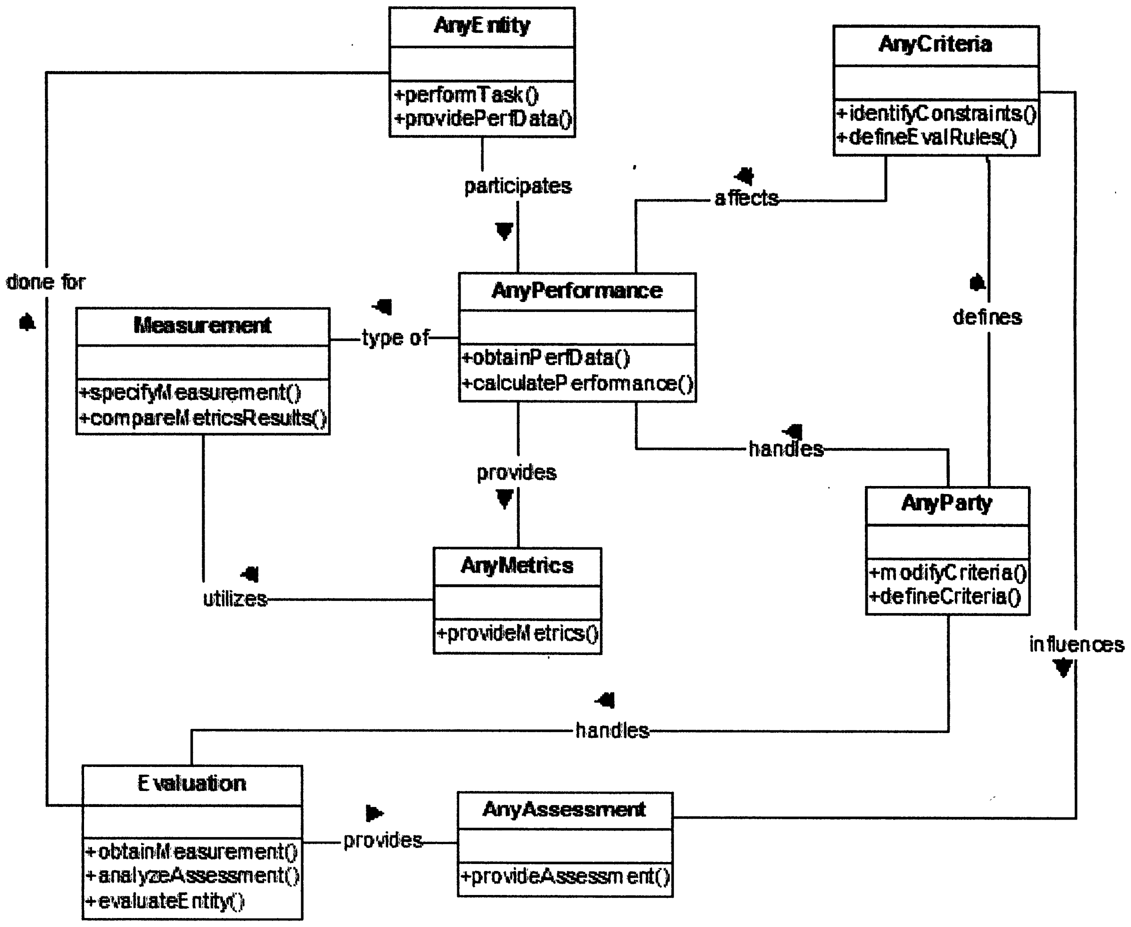


Figure 12: Architectural Pattern for Performance Evaluation

A.5.7 CRC Cards

The CRC cards shown in the following tables provide details on collaboration and responsibilities of various participants in the pattern.

Evaluation (Evaluation facilitator)		
Responsibility	Collaboration	
	Client	Services
Represents a process that attempts to assess the performance of an ongoing or completed activity.	AnyParty, AnyEntity, AnyAssessment, AnyCriteria	defineEvaluation(), analyzeAssessment(), evaluateEntity( )

Table 16: CRC Card for Evaluation

AnyEntity (Evaluation Requestor)		
Responsibility	Collaboration	
Requests performance evaluation to be done based on the criteria and standard set.	Client	Services
	Evaluation AnyPerformance	requestEvaluation(), provideData()

Table 17: CRC Card for AnyEntity

AnyParty (Evaluator)		
Responsibility	Collaboration	
	Client	Services
Setting the rules, along with identifying the criteria, under which performance evaluation of any entity is done.	Evaluation, AnyCriteria	identifyConstraint(), defineEvalRules()

Table 18: CRC Card for AnyParty

AnyCriteria (Criteria Definer)		
Responsibility	Collaboration	
	Client	Server
Describes the rules, standards and the external factors that need to be considered before any task can be performed.	Evaluation, AnyParty	defineConstraint() modifyConstraint()

Table 19: CRC Card for AnyCriteria

AnyAssessment(Assessment Provider)		
Responsibility	Collaboration	
	Client	Server
Provides the status, findings, evidence, and result of any evaluation process.	Evaluation, AnyEntity, AnyConstraints	provideAssessment ( )

Table 20: CRC Card for AnyAssessment

AnyMetrics (Metric Provider)		
Responsibility	Collaboration	
	Client	Server
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.	AnyPerformance, AnyEntity, Measurement	provideMetrics ( )

Table 21: CRC Card for AnyMetrics

A.5.8 Applicability with illustrated examples

Problem Description

The pattern can be used for evaluation of the task of transmission of data in networks, speed of computer processing, order completion status, etc.

Problem Class Diagram

The diagram below depicts the class diagram for the implementation of ***Performance Evaluation*** pattern in the wireless field. The stable parts of the system, i.e., the EBTs and BOs are shown in different columns of the table.

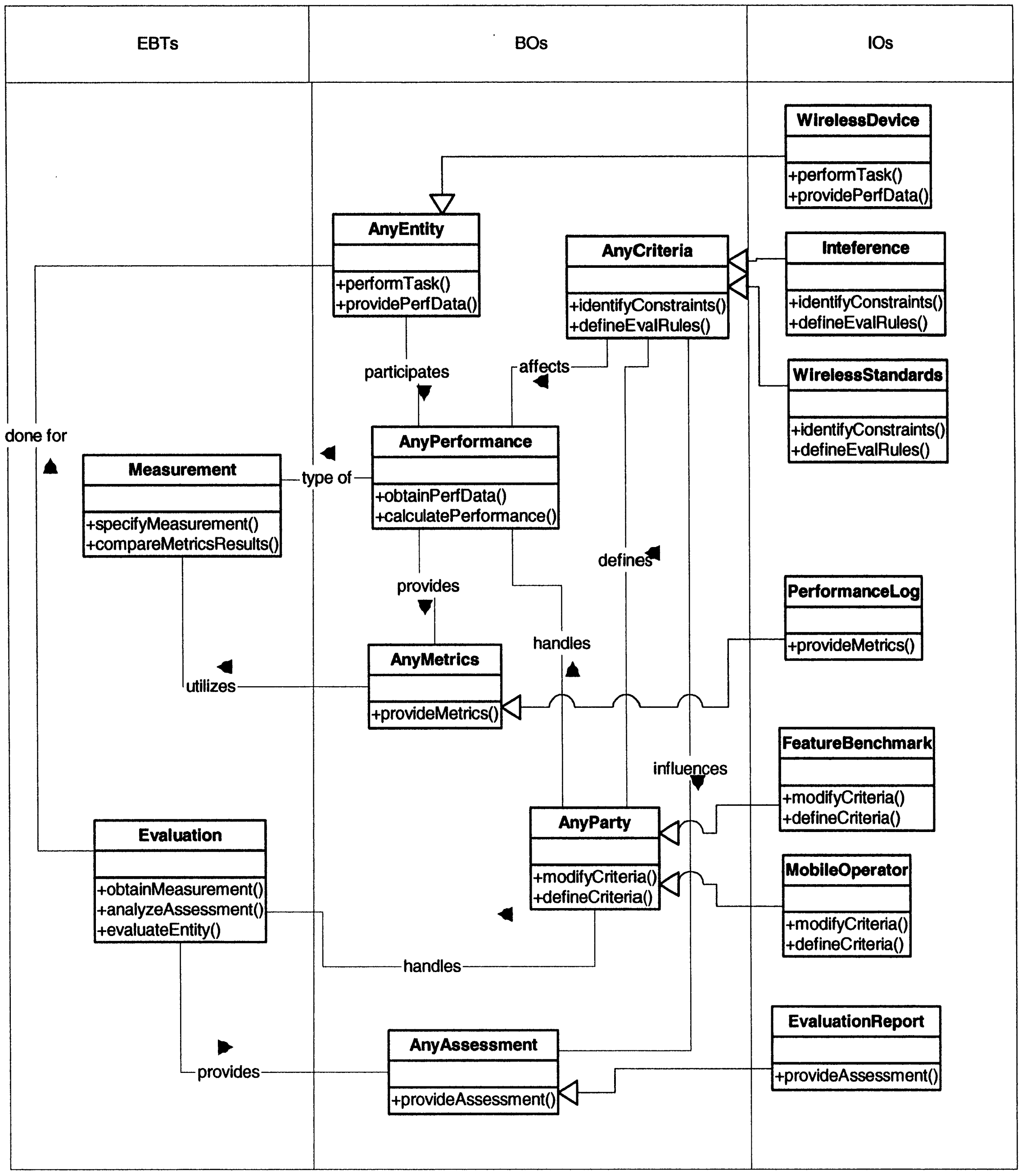


Figure 13: Performance Evaluation Class Diagram for Wireless Devices

The figure below shows the class diagram for the implementation of performance evaluation class diagram for evaluating the performance of a computer in terms of processing of instructions

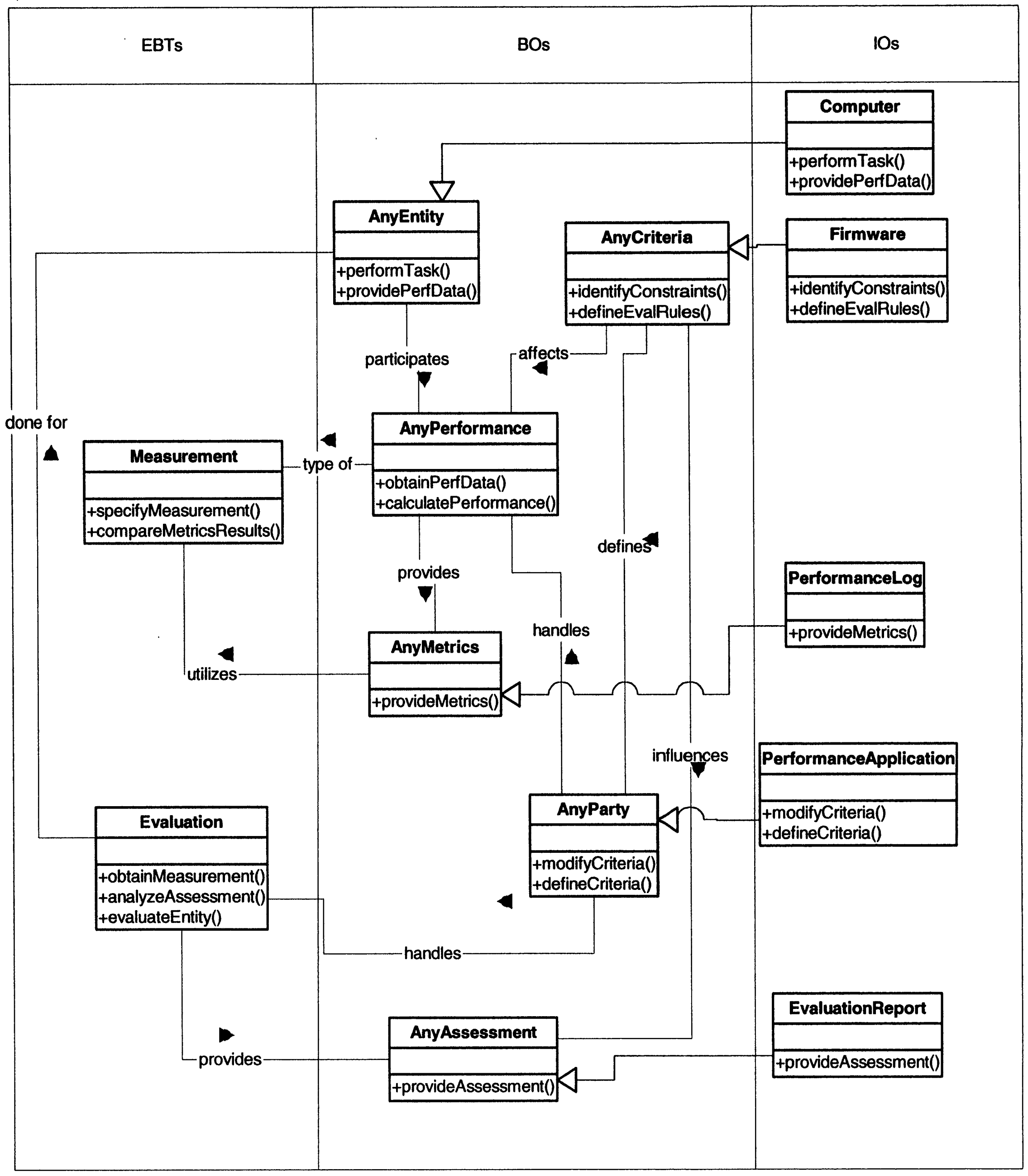


Figure 14: Performance Evaluation Class Diagram for Computer Instruction Processing

A.5.9 Related Patterns

There are several patterns that usually interact with the above described analysis pattern. They are usually classified as:

Related Analysis Patterns

- ***Collection***: This represents the collection of any data that can be obtained during the process of evaluation. It includes all the relevant information that can be used for any analysis that needs to be done once a task is performed and evaluated.
- ***Analysis***: The pattern includes all the analysis techniques used for evaluation. This can include all the processes that perform analysis of the tasks that are evaluated.

A.5.10 Design/ Implementation Issues

Hooks Description

This section provides the description on various hooks that can be attached to this pattern.

- ***AnyParty***: The party that identifies an evaluator can be extended, by attaching an IO. For example, the party or an evaluator for a wireless network can be an operator who can change over a period of time and get replaced by an administrator. For the land evaluation process, the evaluator can be a different party in charge of carrying out the evaluation task. Thus, the party can be in any domain without requiring the entire code to be rewritten for a particular domain.
- ***AnyAssessment***: This BO can have a hook attached to it to encompass different assessments that can be associated with that domain. For example, in a network scenario, the assessments can change, while the pattern remains stable. That is,

the Industrial Objects (IOs) for assessment change with minimal modifications required to encompass these changes.

- ***AnyConstraints***: This can have hooks attached to it to handle external factors and the constraints that affect the evaluation process. These can be considered IOs that can be replaced or modified without affecting the structure of the overall system. Because it can also represent standards that can be modified or updated based on the requirements of a domain, hooks can be provided to handle these changes.

APPENDIX B: APPLICATION SCENARIOS FOR PERFORMANCE PATTERN

B.1 Order Processing and Quality Inspection in Manufacturing Industry

This section provides an explanation of how stable performance design pattern can be applied to the domain of a manufacturing industry to keep track of an order and inspection process. Both of these processes are based on certain standards and criteria. For example, the criteria for the order processing would be the deadline specified by the dealer and the criteria for inspection, the standard specified by the industry for the inspection process. The following section provides the details for each use case.

B.1.1 Use Case Description

Use Case Id: 1.0

Title: Check Order Processing

Actors	Roles
AnyParty	Dealer
AnyEntity	OrderProcessingSystem

Table 22: Actors and Roles for Use Case 1.0

Class	Layer	Super Class	Attributes	Operations

Measurement	EBT		measurementId
			measurementType
				specifyMeasurement( )
				compareMetricsResults( )

AnyPerformance	BO		taskId
			taskType
				evaluateTask( )
				providePerformanceData( )
				analyzeEntityPerformance()
				performanceResult( )
				calculatePerformance ( )

AnyParty	BO		partyId partyName partyType	evalEntityPerformance ( ) identitfyCriteria( )

AnyEntity	BO		entityId
			entityName
				performTask( )
				providePerformanceData( )

			entityType	

AnyCriteria	BO		criteriaId
			criteriaType
			crietriaDescription
				defineCriteria()
				modifyCriteria( )

AnyMetrics	BO		metricsType metricsId	provideMetrics ( )

Table 23: Classes with Attributes and Methods for Use Case 1.0

Description:

1. The dealer, who is AnyParty, sends a request to the AnyPerformance to obtain the status of the order placed based on the taskId or the orderId.
2. AnyPerformance sends a request to AnyCriteria to check for any conditions specified for that task. In this case, the deadline or the delivery date specified by the dealer.
3. The result of the request is then sent back by AnyCriteria to AnyPerformance, thus providing details on the conditions specified by the dealer.
4. To check the status of the task of order completion, AnyPerformance then forwards the request to AnyEntity, which is the OrderProcessingSystem, to check the status of the order placed based on the taskId.
5. AnyEntity, which is the OrderProcessingSystem, then checks if the task specified in the request is completed or is in the process of completion.
6. The result of the task, which is the order in this case, is sent back to AnyPerformance.
7. AnyPerformance then compares the result sent by AnyEntity based on the criteria sent by AnyCriteria.

8. The results obtained from the comparison are then sent to AnyMetrics.
9. Results of the updated process are then sent back to AnyPerformance from AnyMetrics.
10. AnyPerformance then forwards the result of the order status to AnyParty, who is the Dealer who requested the status of the order placed.

Use Case Id: 2.0

Title: Inspect Product

Actors	Roles
AnyParty	QualityInspector
AnyEntity	OrderProcessingSystem

Table 24: Actors and Roles for Use Case 2.0

Class	Layer	Super Class	Attributes	Operations

Measurement	EBT		measurementId
			measurementType
				specifyMeasurement( )
				compareMetricsResults( )

AnyPerformance	BO		taskId
			taskType
				evaluateTask( )
				providePerformanceData( )
				analyzeEntityPerformance()
				performanceResult( )
				calculatePerformance ( )

AnyParty	BO		partyId partyName partyType	evalEntityPerformance ( ) identitfyCriteria( )

AnyEntity	BO		entityId
			entityName
			entityType
				performTask( )
				providePerformanceData( )

AnyCriteria	BO		criteriaId
			criteriaType
			crietriaDescription
				defineCriteria()
				modifyCriteria( )

AnyMetrics	BO		metricsType metricsId	provideMetrics ( )

Table 25: Classes with Attributes and Methods for Use Case 2.0

Description:

1. AnyParty, who is the QualityInspector, sends a request to the AnyPerformance to obtain the quality of the product or order placed based on the taskId or the orderId.
2. AnyPerformance sends a request to AnyCriteria to check for the standards specified for the quality of the product and the conditions specified by the dealer who places the order. For example, the dealer may specify the color and the material to be used for the production of the order. Some standard body that takes care of the overall quality assurance in the industry may specify the standards for the same product.
3. The result of the request is then sent back by AnyCriteria to AnyPerformance, thus providing details on the conditions specified by the dealer along with the standards that need to be met for the product or the order to be accepted.
4. To check for the quality of the product, which has the task of quality assurance for the order, AnyPerformance forwards the request to AnyEntity, which is the OrderProcessingSystem, to obtain the details of the order placed based on the taskId.
5. The OrderProcessingSystem, which is the AnyEntity, then provides the order details of the orderId specified in the request.
6. The details of the order based on the orderId specified, is sent back to AnyPerformance.

7. AnyPerformance then checks for the quality of the product based on the specifications provided by the dealer and the standards, which are sent by AnyCriteria that need to be considered for the quality assurance of the product.
8. The results obtained from the check are then sent to AnyMetrics for the updating the inspection results.
9. Results of the updating the quality results of the product in AnyMetrics are then sent back to AnyPerformance from AnyMetrics.
10. AnyPerformance then forwards the result of the inspection process based on the taskId to AnyParty, who is the Inspector who requested the quality of the order placed.

B.1.2 Sequence Diagram

Check Order Processing

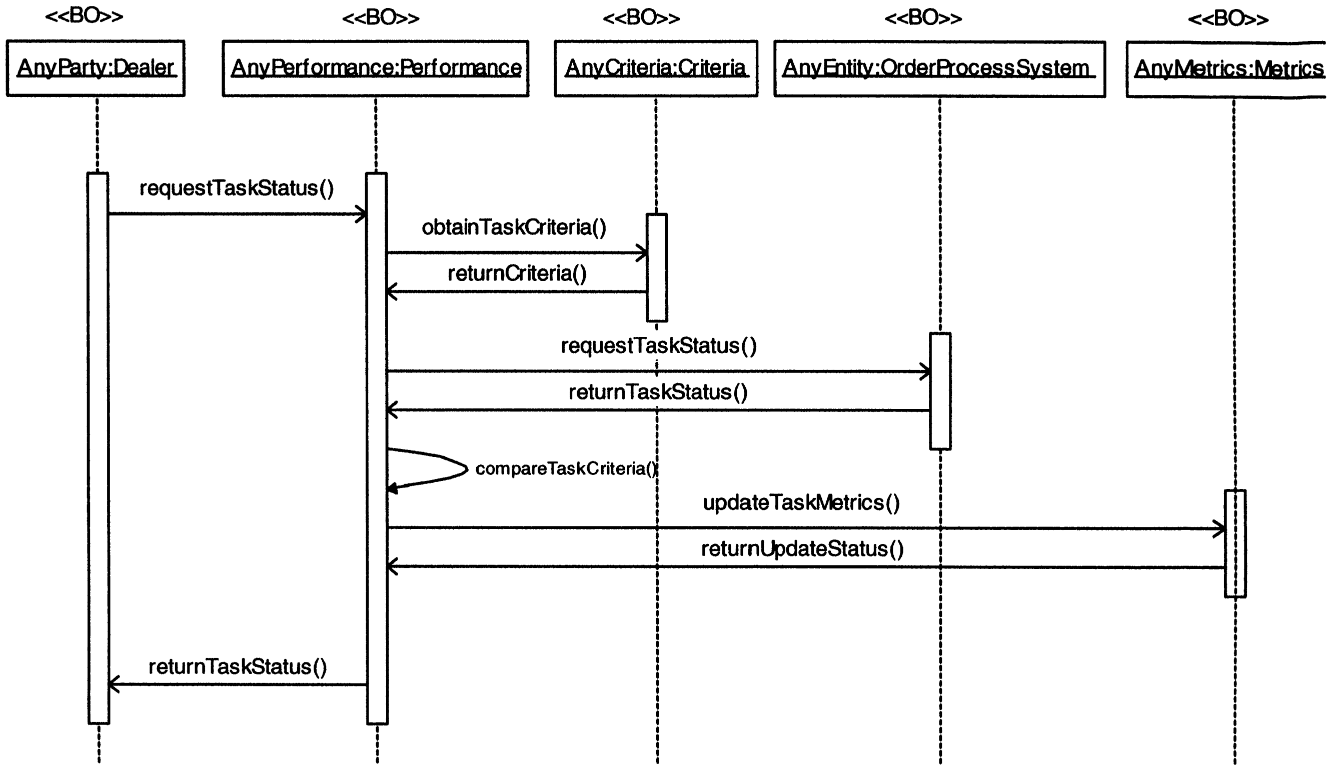


Figure 15: Sequence Diagram for Checking Order Processing

1. The dealer, who is AnyParty, sends a request to the AnyPerformance to obtain the status of the order placed based on the taskId or the orderId with the message requestTaskStatus( ).
2. On receiving the above message, AnyPerformance sends a request to AnyCriteria, obtainTaskCriteria( ) to check for any conditions specified for that task. In this case, AnyPerformance needs the deadline or the delivery date specified by the dealer.

3. The result of the request, which is returnCriteria( ), is sent back by AnyCriteria to AnyPerformance; providing details on the conditions specified by the dealer.
4. To check the status of the task of order completion, AnyPerformance then forwards the request to AnyEntity, which is the OrderProcessingSystem, to check the status of the order placed based on the taskId.
5. AnyEntity, which is the OrderProcessingSystem, then checks if the task specified in the request is completed or is in the process of completion.
6. The result of the task, which is the order in this case, is sent back to AnyPerformance with the message returnTaskStatus( ), which provides the task status results.
7. AnyPerformance then compares the result sent by AnyEntity based on the criteria sent by AnyCriteria which triggers the function compareTaskCriteria( ).
8. The results obtained from the comparison are then sent to AnyMetrics to update the data or the details related to the task present in AnyMetrics, with the message updateTaskMetrics( ).
9. After the details related to the task based on the taskId are updated, the results of the updated process are then back to AnyPerformance from AnyMetrics with returnUpdateStatus( ).
10. AnyPerformance then forwards the result of the order status to AnyParty, who is the Dealer that requested the status of the order placed with returnTaskStatus( ).

Inspect Product

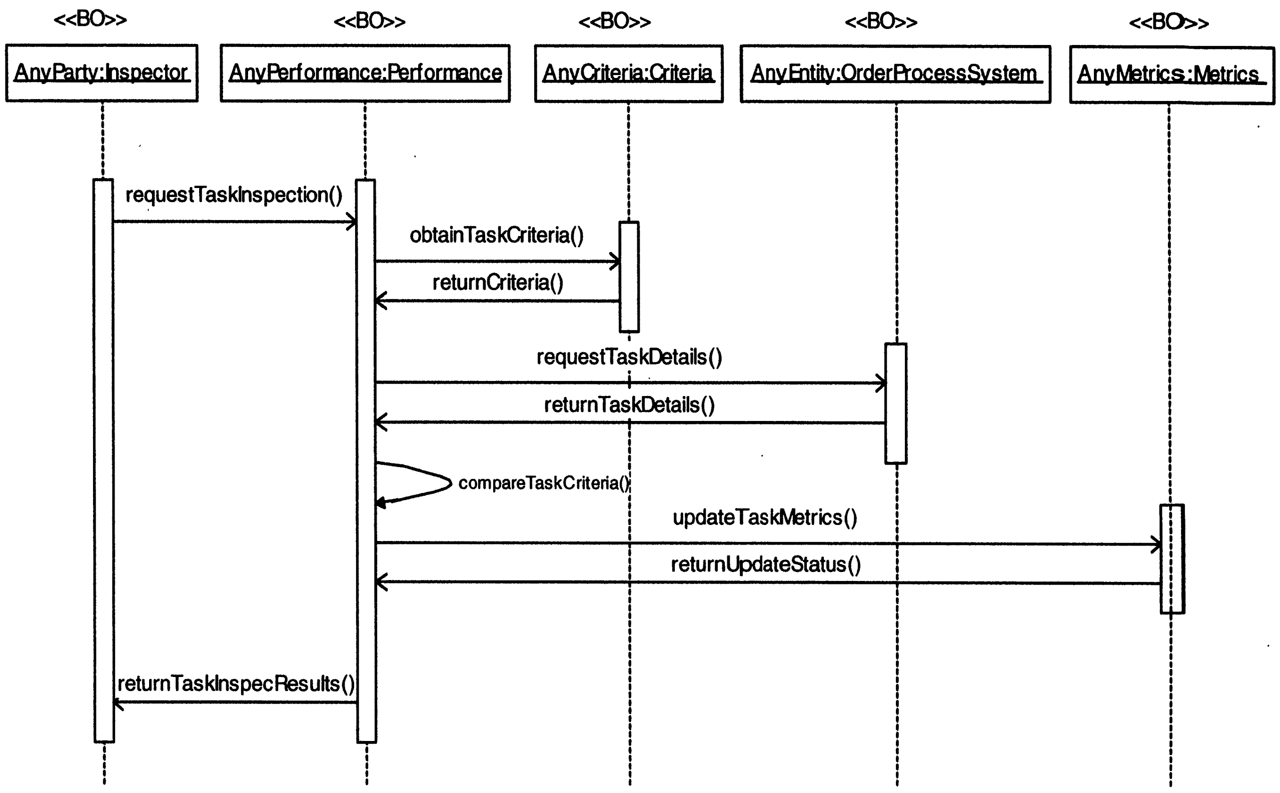


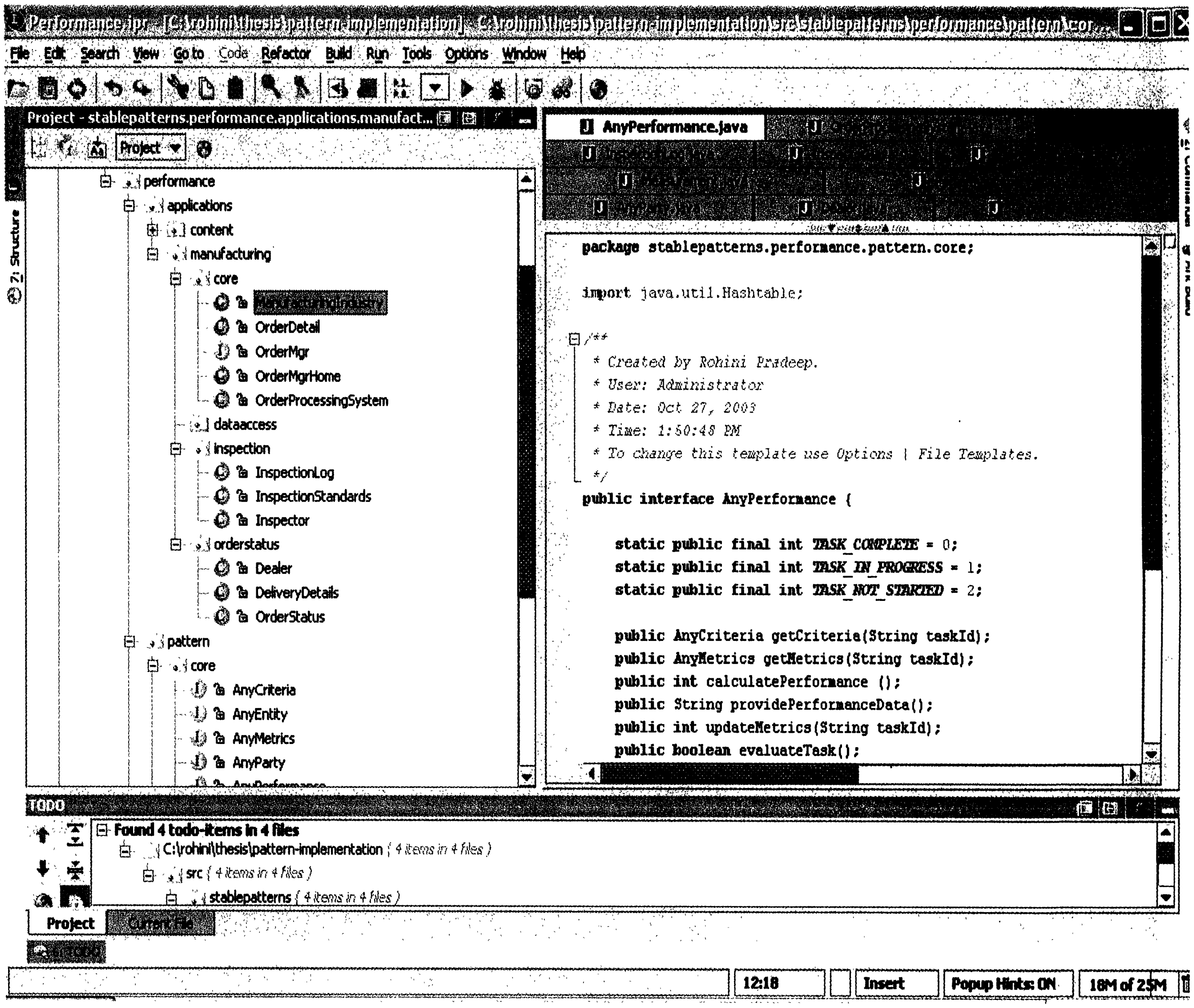
Figure 16: Sequence Diagram to Inspect Product

1. A message, requestTaskInspection( ), requesting the product inspection is sent from the QualityInspector, who is AnyParty, to AnyPerformance to obtain the quality of the product or order placed based on the taskId or the orderId.
2. AnyPerformance sends a request to AnyCriteria to check for the standards specified for the quality of the product and the conditions specified by the dealer who places the order. For example, the dealer may specify the color and the material to be used for the production of the order. Some standard body that takes

care of the overall quality assurance in the industry may specify the standards for the same product.

3. The result of the request is then sent back by AnyCriteria to AnyPerformance, thus providing details on the conditions specified by the dealer along with the standards that need to be met for the product or the order to be accepted.
4. To check for the quality of the product, which has the task of quality assurance for the order, AnyPerformance forwards the request to AnyEntity, which is the OrderProcessingSystem, to obtain the details of the order placed based on the taskId.
5. The OrderProcessingSystem, which is the AnyEntity, then provides the order details of the orderId specified in the request.
6. The details of the order based on the orderId specified, is sent back to AnyPerformance.
7. AnyPerformance then checks for the quality of the product based on the specifications provided by the dealer and the standards, which are sent by AnyCriteria that need to be considered for the quality assurance of the product.
8. The results obtained from the check are then sent to AnyMetrics for the updating the inspection results.
9. The results of the updating the quality results of the product in AnyMetrics are then sent back to AnyPerformance from AnyMetrics.
10. AnyPerformance then forwards the result of the inspection process based on the taskId to AnyParty, the Inspector who requested the quality of the order placed.

B.1.3 Package Structure


File Edit Search View Go to Code Refactor Build Run Tools Options Window Help
Project - stablepatterns.performance.applications.manufact...
Project
AnyPerformance.java
7: Structure
TODO
Project
Current File
12:18
Insert
Popup Hints: ON
18M of 25M

```
performance
  applications
    content
    manufacturing
      core
        ManufacturingIndustry
        OrderDetail
        OrderMgr
        OrderMgrHome
        OrderProcessingSystem
      dataaccess
      inspection
        InspectionLog
        InspectionStandards
        Inspector
      orderstatus
        Dealer
        DeliveryDetails
        OrderStatus
  pattern
    core
      AnyCriteria
      AnyEntity
      AnyMetrics
      AnyParty
```

```
package stablepatterns.performance.pattern.core;

import java.util.Hashtable;

/**
 * Created by Rohini Pradeep.
 * User: Administrator
 * Date: Oct 27, 2003
 * Time: 1:50:48 PM
 * To change this template use Options | File Templates.
 */
public interface AnyPerformance {

    static public final int TASK_COMPLETE = 0;
    static public final int TASK_IN_PROGRESS = 1;
    static public final int TASK_NOT_STARTED = 2;

    public AnyCriteria getCriteria(String taskId);
    public AnyMetrics getMetrics(String taskId);
    public int calculatePerformance ();
    public String providePerformanceData();
    public int updateMetrics(String taskId);
    public boolean evaluateTask();
```

```
Found 4 todo-items in 4 files
  C:\rohini\thesis\pattern-implementation ( 4 items in 4 files )
    src ( 4 items in 4 files )
      stablepatterns ( 4 items in 4 files )
```

Figure 17: Package Structure for Performance Pattern Implementation in Manufacturing Industry

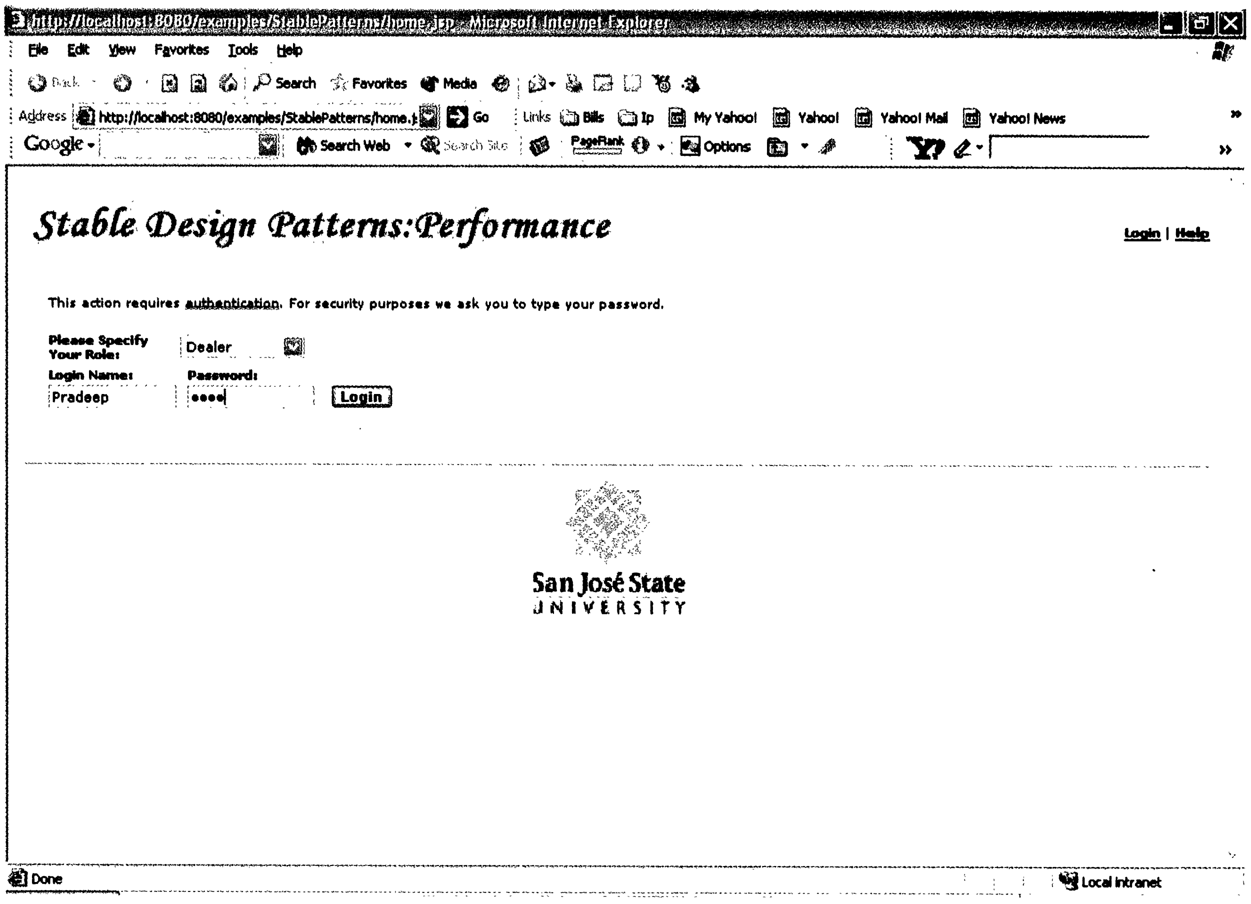
The figure above depicts the package structure used for developing application for performance pattern. The classes for the pattern are put under "pattern", then "core" directory. Classes related to different applications are put under relevant directories. The next section shows a series of screen shots for the performance pattern, when used in manufacturing industry for checking the order status and inspection processes.

Sample Screen Shots for the application: Manufacturing Industry

The Figure 4 shows the first screen that is displayed to the user during login to either:

1) Login as a dealer to check the status of the order placed or

2) Login as an inspector to carry out the inspection process.



http://localhost:8080/examples/StablePatterns/home.jsp - Microsoft Internet Explorer

Stable Design Patterns:Performance

Login | Help

This action requires authentication. For security purposes we ask you to type your password.

Please Specify Your Role: Dealer

Login Name: Pradeep

Password: ●●●●

Login

San José State University

Figure 18: Screen Shot for Login Screen in Manufacturing Industry

The user has the option of either choosing the role of a "Dealer" or "Inspector." The "login" button is chosen after the user enters the login name and the password.

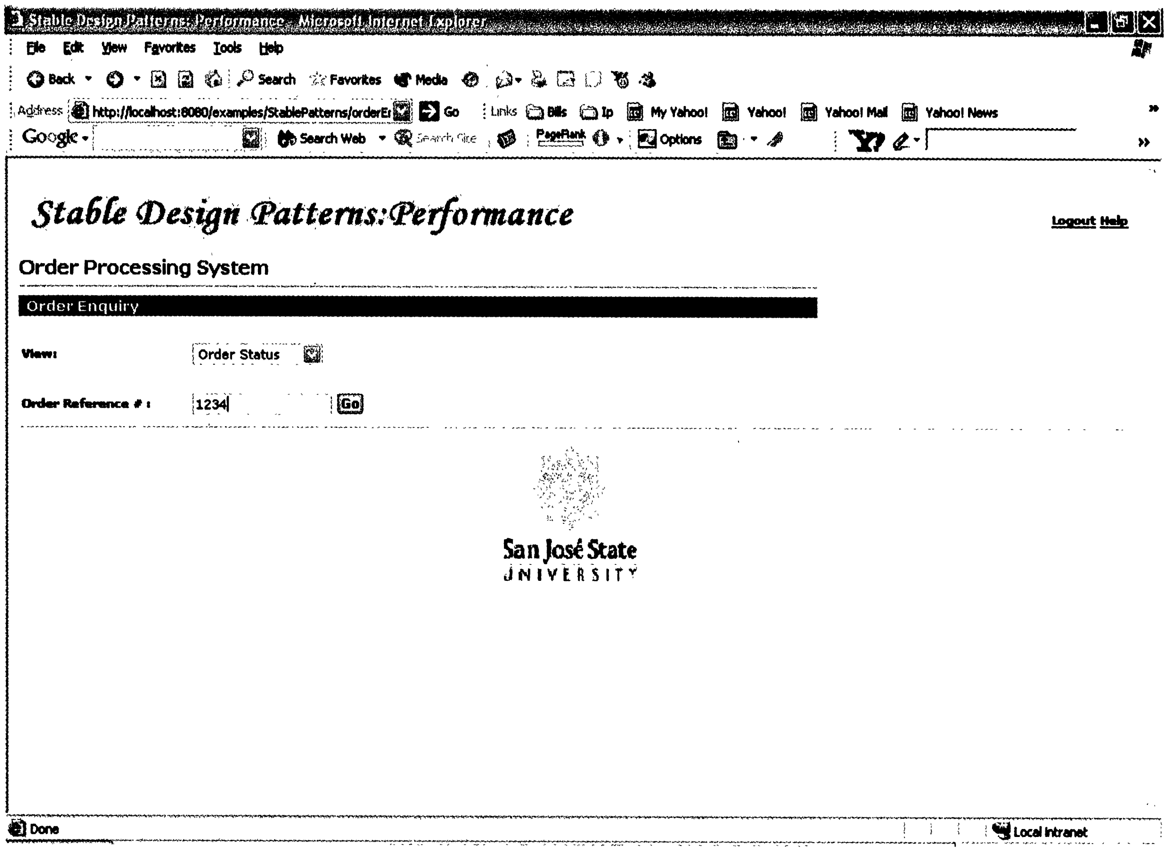
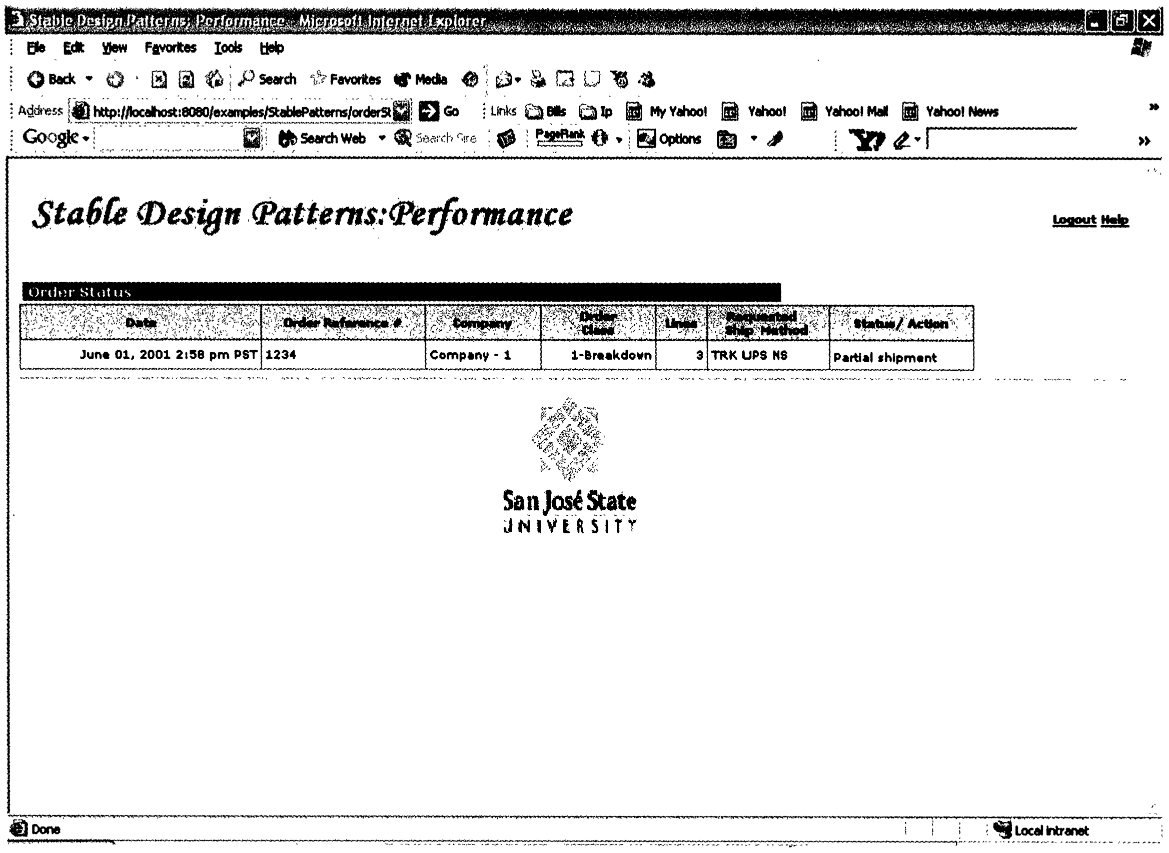

Stable Design Patterns: Performance - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:8080/examples/StablePatterns/orderE
Stable Design Patterns:Performance
Logout Help
Order Processing System
Order Enquiry
View:
Order Status
Order Reference # :
1234
Go
San José State University
Done
Local intranet

Figure 19: Screen Shot for the Order Processing System

Once the user logs in as a dealer, the order processing system is displayed. The order status of a particular product based on the order reference number can be checked. The dealer enters the order reference number and clicks the "Go" button after which the screen with the order details is displayed. The next screen shot describes this.


Stable Design Patterns: Performance - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/examples/StablePatterns/orderSt Go Links Bills Ip My Yahoo! Yahoo! Yahoo! Mail Yahoo! News
Google Search Web Search Site PageRank Options
Logout Help
San José State University
Done
Local intranet

Stable Design Patterns:Performance

Order Status

Date	Order Reference #	Company	Order Class	Lines	Requested Ship Method	Status/ Action
June 01, 2001 2:58 pm PST	1234	Company - 1	1-Breakdown	3	TRK UPS NS	Partial shipment

Figure 20: Screen Shot for the Order Details

The figure above displays the screen that provides the user with the details of the order placed. This is the result of entering a valid order reference number. This screen uses the information from the performance class, which interacts with the metrics class to provide the relevant details.

B.2 Performance of a Game Application and Animation in Wireless Devices

This section provides an illustration of how stable performance design pattern can be applied wireless devices using screen shots in different use cases. The cases of checking the performance of running a typical game in a benchmark and performance of animation in wireless devices are considered. The following section provides the details for each use case.

C.2.1 Use Case Description

Use Case Id: 1.0

Title: Check Game Performance

Actors	Roles
AnyParty	GameBenchmark
AnyEntity	GameMidlet

Table 26: Actors and Roles for Usecase1.0

Class	Layer	Super Class	Attributes	Operations

Measurement	EBT		measurementId
			measurementType
				specifyMeasurement( )
				compareMetricsResults( )

AnyPerformance	BO		taskId
			taskType
				evaluateTask( )
				providePerformanceData( )
				analyzeEntityPerformance()
				performanceResult( )
				calculatePerformance ( )

AnyParty	BO		partyId partyName partyType	evalEntityPerformance ( ) identitfyCriteria( )

AnyEntity	BO		entityId
			entityName
			entityType
				performTask( )
				providePerformanceData( )

AnyCriteria	BO		criteriaId
			criteriaType
			crietriaDescription
				defineCriteria()
				modifyCriteria( )

AnyMetrics	BO		metricsType metricsId	provideMetrics ( )

Table 27: Classes with Attributes and Methods for Use Case 1.0

Description:

1. The GameBenchmark, which is AnyParty, sends a request to the AnyPerformance to obtain data related to the performance of the game application.
2. AnyPerformance sends a request to AnyCriteria to check for any conditions specified for that game application. For example, it can be the acceptable threshold value for running that game in an ideal situation.
3. The result of the request is then sent back by AnyCriteria to AnyPerformance, thus providing details on the conditions to run the application.
4. To begin capturing the required data, AnyPerformance then forwards the request to AnyEntity, which is the game MIDlet, to start running the game application.
5. AnyEntity, which is the MIDlet, then starts a timer to compute the run time for running the application.
6. Once the game is run to completion, which is based on a typical scenario of running that game, the computed value of running that application is sent to AnyPerformance.
7. AnyPerformance then compares the result sent by AnyEntity based on the criteria sent by AnyCriteria.
8. The results obtained from the comparison are then sent to AnyMetrics.
9. Results of the updated process are then sent back to AnyPerformance from AnyMetrics.

10. AnyPerformance then forwards the result of the performance of game application to AnyParty, which is the GameBenchmark that requested that particular data.

Use Case Id: 2.0

Title: Animation Performance

Actors	Roles
AnyParty	FeatureBenchmark
AnyEntity	AnimationMIDlet

Table 28: Actors and Roles for Use Case 2.0

Class	Layer	Super Class	Attributes	Operations

Measurement	EBT		measurementId
			measurementType
				specifyMeasurement( )
				compareMetricsResults( )

AnyPerformance	BO		taskId
			taskType
				evaluateTask( )
				providePerformanceData( )
				analyzeEntityPerformance()
				performanceResult( )
				calculatePerformance ( )

AnyParty	BO		partyId partyName partyType	evalEntityPerformance ( ) identitfyCriteria( )

AnyEntity	BO		entityId
			entityName
			entityType
				performTask( )
				providePerformanceData( )

AnyCriteria	BO		criteriaId
			criteriaType
			crietriaDescription
				defineCriteria()
				modifyCriteria( )

AnyMetrics	BO		metricsType metricsId	provideMetrics ( )

Table 29: Classes with Attributes and Methods for Use Case 2.0

Description:

1. AnyParty, which is the FeatureBenchmark, sends a request to the AnyPerformance to obtain data related to the performance of animation feature in any wireless toolkit.

2. AnyPerformance, which is AnimationPerformance, sends a request to AnyCriteria to check for the threshold value specified for running the animation feature in the ideal scenario.
3. The result of the request is then sent back by AnyCriteria to AnyPerformance, thus providing details on the conditions to run the feature benchmark, animation.
4. To begin capturing the required data, AnyPerformance then forwards the request to AnyEntity, the AnimationMIDlet, to start the animation of any given image.
5. AnyEntity, the AnimationMIDlet, then starts a timer to compute the run time for executing the animation feature benchmark.
6. Once the animation is done for a specified cycle, the computed value of running that feature benchmark is sent to AnyPerformance.
7. AnyPerformance then compares the result sent by AnyEntity based on the criteria sent by AnyCriteria.
8. The results obtained from the comparison are then sent to AnyMetrics.
9. Results of the updated process are then sent back to AnyPerformance from AnyMetrics.
10. AnyPerformance then forwards the result of the performance of the animation feature benchmark to AnyParty, which is the AnimationBenchmark that requested the data.

B.2.2 Sequence Diagram

Check Game Performance

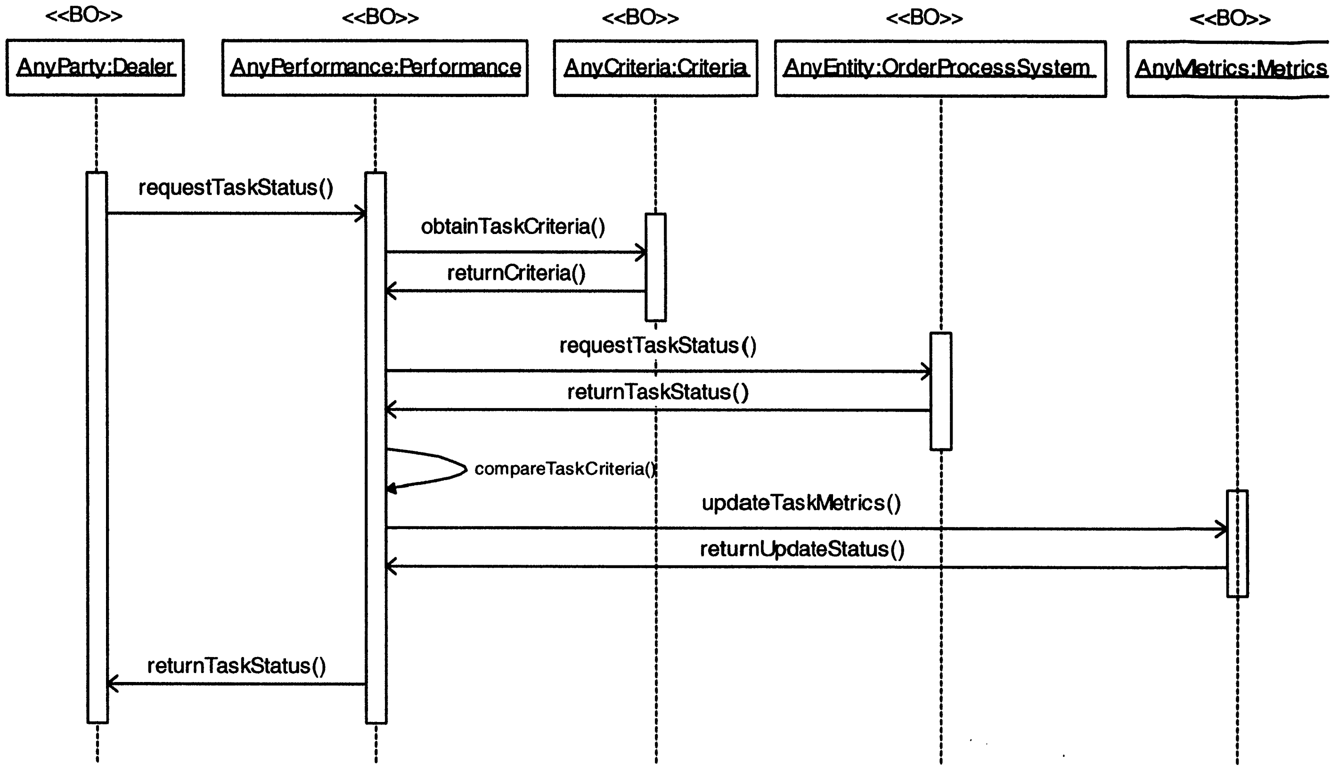
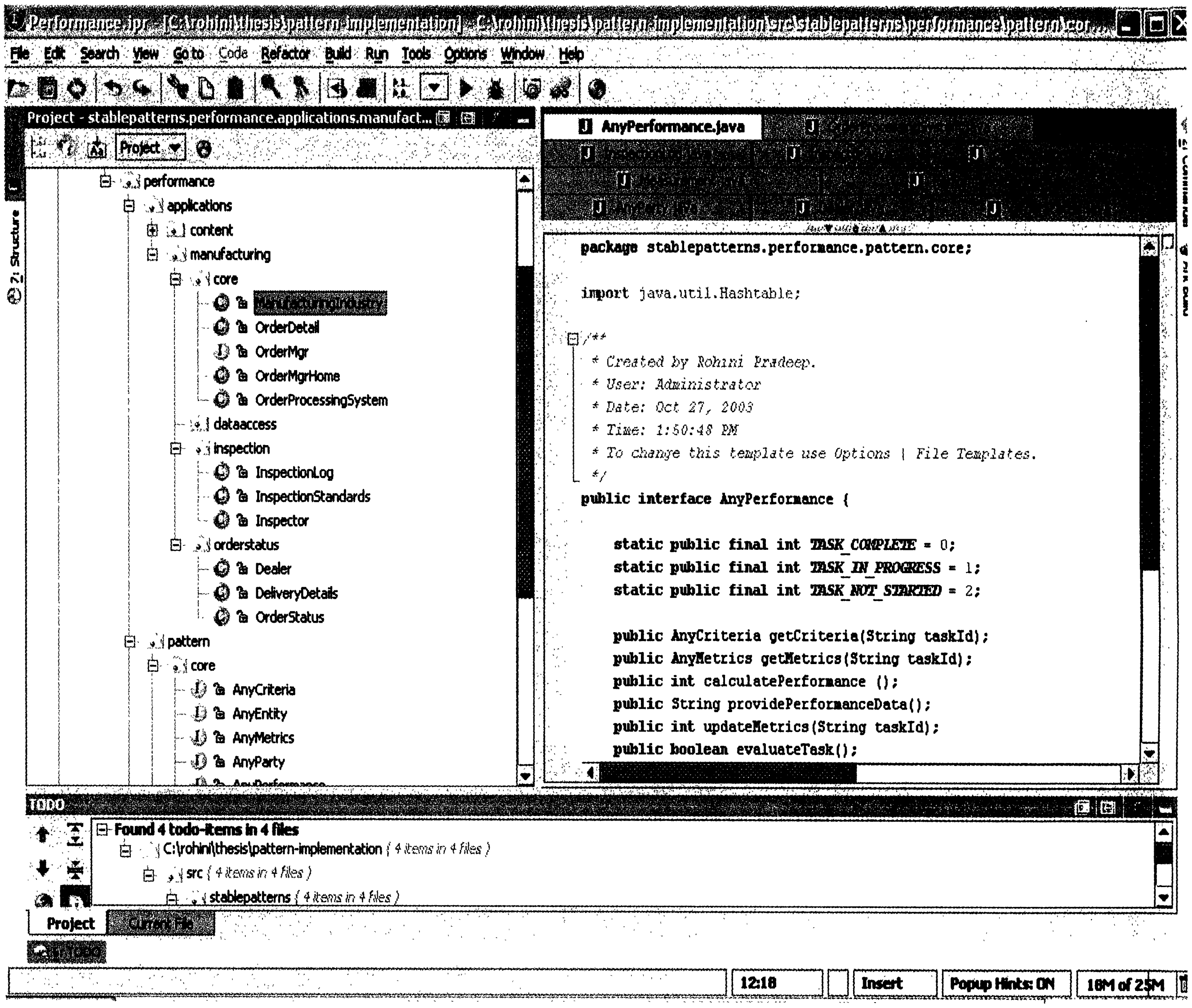


Figure 21: Sequence Diagram for Checking Game Performance

1. The GameBenchmark, which is AnyParty, requests data related to the performance of the game application from AnyPerformance with the message getGamePerformance( ).
2. On receiving the above message, AnyPerformance sends a request to AnyCriteria, getGameCriteria( ) to check for any conditions, like threshold value or ideal game application values specified for the game scenario benchmark.

3. The result of the request, which is returnCriteria( ), is sent back by AnyCriteria to AnyPerformance; providing details on the conditions for running the game application.
4. To check the run time of the game, AnyPerformance then forwards the request to AnyEntity, which is the GameMIDlet, to run the application.
5. The game application is run, during which the startup time and the wrap up time is stored.
6. The result of the game application run time is sent back to AnyPerformance with the message returnGamePerformance( ).
7. AnyPerformance then compares the result sent by AnyEntity, and the criteria sent by AnyCriteria which triggers the function compareGameCriteria( ).
8. The results obtained from the comparison are then sent to AnyMetrics to update the data or the details related to the task present in AnyMetrics, with the message updateGameMetrics( ).
9. After the details related to the performance of the game application are updated, the results of the updated process are sent back to AnyPerformance from AnyMetrics with returnUpdateStatus( ).
10. AnyPerformance then forwards the result of the order status to AnyParty, which is the GameBenchmark that requested the performance time of running a typical game application, with returnGamePerfTime( ).

B.2.3 Package Structure



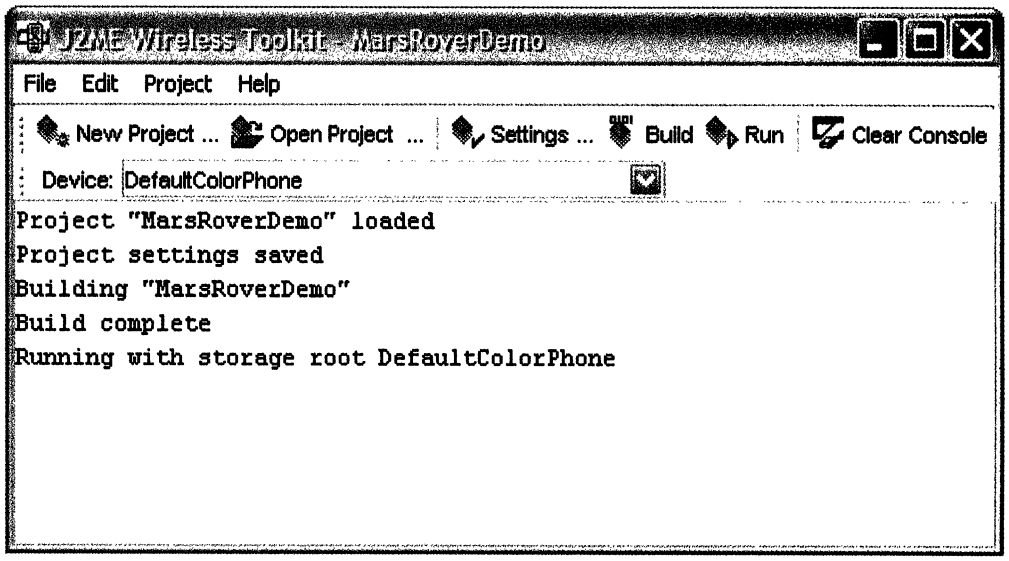
```
package stablepatterns.performance.pattern.core;

import java.util.Hashtable;

/**
 * Created by Rohini Pradeep.
 * User: Administrator
 * Date: Oct 27, 2003
 * Time: 1:50:48 PM
 * To change this template use Options | File Templates.
 */
public interface AnyPerformance {

    static public final int TASK_COMPLETE = 0;
    static public final int TASK_IN_PROGRESS = 1;
    static public final int TASK_NOT_STARTED = 2;

    public AnyCriteria getCriteria(String taskId);
    public AnyMetrics getMetrics(String taskId);
    public int calculatePerformance ();
    public String providePerformanceData();
    public int updateMetrics(String taskId);
    public boolean evaluateTask();
```

Figure 22: Package Structure for Performance Pattern Implementation in Game Application

The figure above depicts the package structure used for developing application for performance pattern. The classes for the pattern are put under "pattern", then "core" directory. Classes related to different applications are put under relevant directories. The next section shows a series of screen shots for the performance pattern, when used in checking the performance of running a typical game application.

Sample Screen Shots for the application: Game Application Performance

The Figure 10 shows the screen when the wireless toolkit that is used to run the game application is used. The desired project is loaded and "Run" is selected to run the MarsRoverGame application.


J2ME Wireless Toolkit - MarsRoverDemo
File Edit Project Help
New Project ... Open Project ... Settings ... Build Run Clear Console
Device: DefaultColorPhone

```
Project "MarsRoverDemo" loaded
Project settings saved
Building "MarsRoverDemo"
Build complete
Running with storage root DefaultColorPhone
```

Figure 23: Screen Shot for Loading the Game Application Using a Wireless Toolkit

The wireless toolkit starts a default color phone to start the game application. The next screen gives the details of the simulated wireless device.

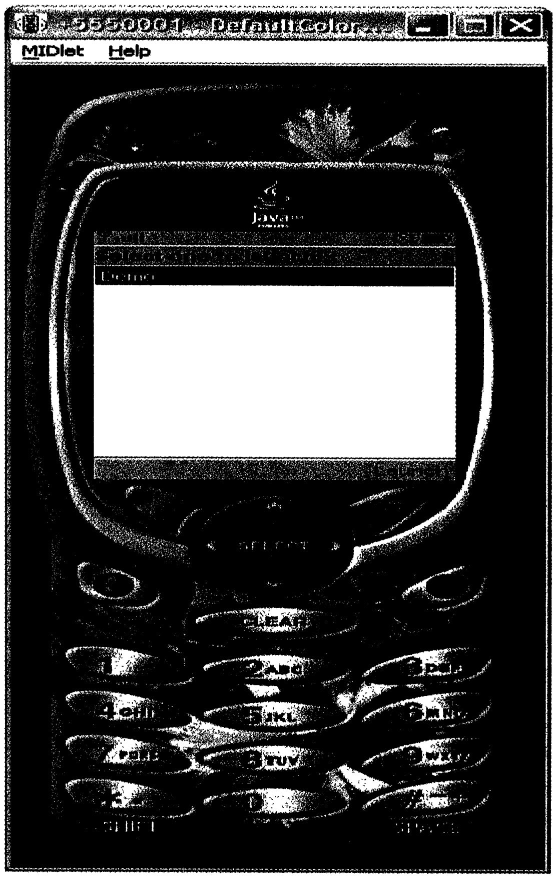


Figure 24: Screen Shot for Choosing the Game Application

Once the game is displayed on the phone screen, the user can start the game using the "Launch" button provided on the wireless device. The next screen displays the game application running.

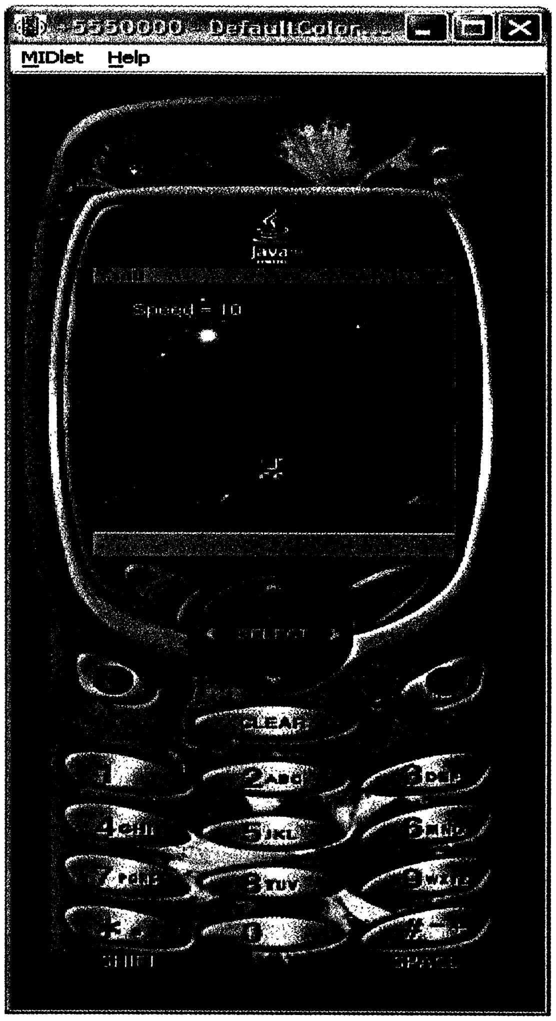
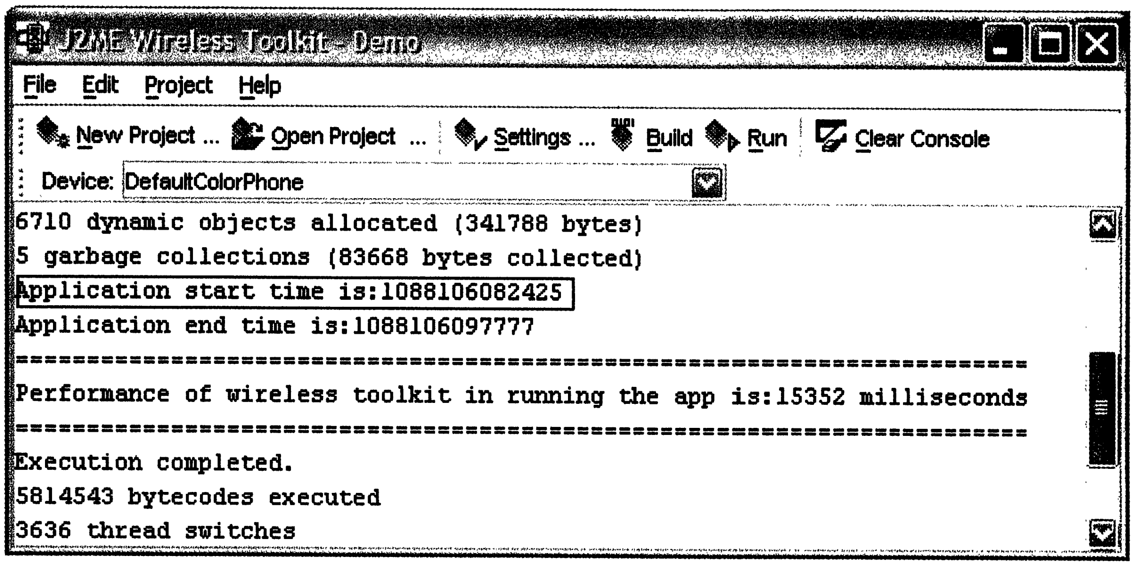


Figure 25: Screen Shot for the Game in Progress

The figure above displays a typical game scenario running on a wireless device. The relevant classes in the BOs and IOs are triggered to handle the data obtained during runtime.


J2ME Wireless Toolkit - Demo
File Edit Project Help
New Project ... Open Project ... Settings ... Build Run Clear Console
Device: DefaultColorPhone

```
6710 dynamic objects allocated (341788 bytes)
5 garbage collections (83668 bytes collected)
Application start time is:1088106082425
Application end time is:1088106097777
========================================================================
Performance of wireless toolkit in running the app is:15352 milliseconds
========================================================================
Execution completed.
5814543 bytecodes executed
3636 thread switches
```

Figure 26: Performance Results of Running a Typical Game Application

Figure 12 shows the results of running the game application in milliseconds.

APPENDIX C: CODE SAMPLES

The code sample below shows the implementation of the IO for AnyEntity, i.e., the MarsRoverDemo. This is used for running the game application on a wireless toolkit.

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.lcdui.Display;
/**
 * Created by IntelliJ IDEA.
 * User: Rohini
 * Date: Jun 11, 2004
 * Time: 10:52:41 AM
 * To change this template use Options | File Templates.
 */
public class MarsRoverDemo extends MIDlet implements Runnable,
AnyEntity{

    private static final int ITERATIONS = 1200;
    private Display display;
    private MarsRoverCanvas marsRoverCanvas;

    private String backGroundImagePath;
    private String tilesImagePath;

    private int loopCount = ITERATIONS;
    private long appStartTime = 0;
    private long appEndtime = 0;
    private long appRunTime = 0;
    private AnyPerformanceImpl perf;

    /**
     * startApp displays a Mars landscape with obstacles
     * and a Mars rover the user can move. A predefined sequence of
     * user input moves the rover across the landscape.<p>
     *
     */
    protected void startApp() throws MIDletStateChangeException {

        appStartTime = System.currentTimeMillis();
        display = Display.getDisplay(this);

        System.out.println("Application start time is:" +
        appStartTime);
        backGroundImagePath = "/background.png";
        tilesImagePath = "/tiles.png";
        try {
                marsRoverCanvas = new
                MarsRoverCanvas(backGroundImagePath, tilesImagePath);
                Thread gameScrollerThread = new Thread(this);
            display.setCurrent(marsRoverCanvas);
        gameScrollerThread.start();
        // start the thread to play the demo
        try {
                  gameScrollerThread.join();
                 // wait till the game iterations complete
        } catch (InterruptedException e) {
        }
        } catch (Exception e) {
        /* This happens if there is a problem while loading Tiles
        */ Image or BackgroundImage.
        }
        try {
          destroyApp(false);
          notifyDestroyed();
        } catch (MIDletStateChangeException ex) {
        }
}

protected void pauseApp() {
}

protected void destroyApp(boolean unconditional) throws
MIDletStateChangeException {
}
/**
 * Advances the game for each iteration.
 * After each iteration, metricUnitFinished(UNIT_FRAME) will be
 * invoked.
 * This thread sleeps 1 milli second for each iteration.
 */
public void run() {
    int lc = 0;
    while (lc++ < loopCount) {

        if(lc >= 400 && lc <= 450) {
            marsRoverCanvas.setDelay((lc /10) % 10); //slow down
        }

        if(lc >= 800 && lc <= 850) {
           marsRoverCanvas.setDelay(5 -((lc / 10) % 10)); //speedup
        }

        if (!marsRoverCanvas.advance()) break;
        //if road is over return
    }
    appEndtime = System.currentTimeMillis();
    appRunTime = appEndtime - appStartTime;
 }
public long getAppRunTime() {
    return appRunTime;
}

}
```