Fall 2015

Consumer Complaints and Protection: Stable Analysis and Design Patterns

Vishnu Sai Reddy Gangireddy
San Jose State University

CONSUMER COMPLAINTS AND PROTECTION: STABLE ANALYSIS AND DESIGN PATTERNS

A Thesis

Presented to

The Faculty of the Department of Computer Engineering

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Vishnu Sai Reddy Gangireddy

December 2015

The Designated Thesis Committee Approves the Thesis Titled

CONSUMER COMPLAINTS AND PROTECTION:
STABLE ANALYSIS AND DESIGN PATTERNS

by

Vishnu Sai Reddy Gangireddy

APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING

SAN JOSÉ STATE UNIVERSITY

December 2015

Dr. M. E. Fayad	Department of Computer Engineering
Dan Harkey	Department of Computer Engineering
Hungwen Li	Department of Computer Engineering

ABSTRACT

CONSUMER COMPLAINTS AND PROTECTION:
STABLE ANALYSIS AND DESIGN PATTERNS

by Vishnu Sai Reddy Gangireddy

The concept of consumer complaints and protection has numerous applications across various domains. Using traditional methods of modeling design patterns is a tedious and costly task. The Software Stability Mode (SSM) is a more efficient and effective modeling method. In this thesis, the differences between the traditional method and the SSM is addressed. Then, several patterns are developed using the SSM to deal with consumer complaints. Each area, Advice, Appraisal, Commitment, Complaint, Compliance, Deed, Guideline, Gratification, Judgment, Model, Need, Ownership, Promotion, Rate, Review, Selling, Support, View, and Violation, is explored and the core knowledge of the concept of consumer complaints and protection is developed visually as well as in detail. Useful SAP and SDP templates are included for each concept. The main contribution of this thesis is the creation of stable, reusable templates that build an unlimited number of applications for the consumer complaints and protection concept.

ACKNOWLEDGEMENTS

I would like to express gratitude to my advisor, Dr. Mohamed E. Fayad, without whom this thesis would not have been possible. The support and guidance that he has provided me throughout my entire journey at San José State University cannot be forgotten.

I would also like to thank my thesis committee members, Prof. Dan Harkey and Prof. Hungwen Li, for their comments and encouragement they have provided to me.

I am grateful to my parents, Lalitha Reddy and Subba Reddy, for all that they have done for me to bring me this far with their best wishes. Also, I am extremely thankful to my sweet sister, Kirthi, and my brother-in-law, Ravi, for supporting me in every aspect of my life.

Last but not least, I would like to thank my roommates and friends for all the parties and outings we had, your company makes me feel lively and positive toward life.

Table of Contents

List of Tables

List of Figures

Chapter 1: Overview

Consumer Complaints and Protection

With the rapid evolution and deep permeation of Internet technology into our lives, today's civil law faces new challenges in safeguarding the rights of a consumer. Consumer rights about a product or services relate to the issues of quantity, quality, distribution, information, service, warranty, price, and consumer viewpoint. Current Internet technology makes a government less prepared to monitor and control the increasing number of products and service providers. Therefore, the importance of a loyal consumer base is a priority for almost all businesses and corporations. Although corporations realize that providing quality service and product delivery is an important part of their business, they often fail to protect and fulfill the expectations of their consumers.

Introduction

Modern day business models face increasing challenges to their marketing strategies regarding consumer satisfaction and protection. The growing influence of e-commerce has made the issue of consumer rights protection even more serious. From a business perspective, it is becoming even more important to build efficient models of trust and reputation. However, the biggest challenge in developing such systems is the fact that trust is mostly considered subjective (Cohen, Regan, & Tran, 2005). Most consumers purchase products or services with certain expectations, and if they fail to find the expectations fulfilled, they are likely to choose a different brand or service. In other words, it is the classic case of buying or using something while not initially knowing its consequences. In such a scenario, if a product is good or service is satisfactory, the consumer remains loyal. But if it turns out to be bad, the consumer is more likely to complain before Finally, moving onto another product or service. Therefore, consumer protection helps drive satisfaction and also enables organizations to reexamine their policies and practices to meet the consumer welfare expectations. Managing such expectations is vital for any organization to retain its consumer base.

Consumer protection has traditionally been a neglected aspect in business operations. This is because not much effort is being put into recognizing and protecting the needs of the consumer base. Most often, the rights of a consumer are either inadequately recognized, or their interests are not properly safeguarded. Furthermore, cost variations also act as an important factor in consumer satisfaction. The more a person pays for a product or service, the greater his or her expectations are concerning those purchases. When considering these factors, the rhetoric about consumer protection is usually found to be based on the reality of consumer dissatisfaction (Agasti & Sengupta, 2014).

Usually, an informal complaint is the most common way of registering one's dissent over a product or service in the hope of getting it resolved amicably. But, if such a grievance is not resolved in a satisfactory manner, the consumer may sometimes even register a formal complaint with a third party such as the Better Business Bureau, a county government, the Federal Trade Commission, etc. However, more often than not, this can be a very cumbersome process. The most common form of registering consumer complaints is either through some specific software application or by recording the complaint in a database with spreadsheets or similar tools. Furthermore, with the advent of the Internet and social media, any impediment to an easy way of expressing dissatisfaction with something is greatly reduced, and a larger number of people have access to the complaints.

Consumer complaints may assume different meanings in different contexts. What works for a specific domain may not work for others. This creates the need for a system that can tackle different application scenarios dealing with consumer complaints. The main goal of this research is to establish the foundations of such a stable system that can work across an umbrella of businesses and application scenarios with the maximum ease of use.

There are several aspects of an online trading application that differ from the traditional ways of consumerism, such as (a) access to product information that is only in the form of digital advertisements rather than physical form; (b) digital forms of payment that may sometimes be tricky and opaque; and (c) authenticating the validity of the contract between the seller and the buyer, which may be complicated. There is a dire need for a system that can efficiently manage the rights of the consumer and at the same time record and maintain complaints dealing with all the aspects of businesses.

Within this thesis, we cover the aforementioned transient aspects of consumer complaints and protection through an amalgamation of different analysis and design patterns built using SSM concepts. These patterns are obtained by analyzing and extracting the core knowledge of the concept of complaints. A system designed using these patterns would simplify the process of modeling different applications that deal with problems related to complaints. The main contribution of the thesis is the creation of stable reusable templates that can be used to build unlimited number of applications for this concept. Furthermore, this work also envisages the creation of several templates for stable analysis and SDPs that can benefit the field of Unified Modeling Language (UML) design immensely.

Problem

A consumer complaint can be understood as a statement of dissatisfaction toward a product or a service. On the other hand, consumer protection is comprised of the laws that give consumers the right to register their dissatisfaction about any abusive or inferior business practices and that get a reasonable resolution. Consumer reports are a collection of data about different products and services through reviews and comparisons. Such reports help in analyzing the good and the bad side of a product or a service. Consumer reports also include research about a product or service to highlight its advantages and disadvantages. However, such reports are usually limited in scope and applicability. Most of these reports are based on results from in-house laboratory tests and experiments. This limits consumers from submitting their reviews or complaints about their purchases. Even the reporting agencies that allow a consumer to submit his or her concerns usually do not collate the user feedback in a meaningful way.

Although the concept of consumer complaints and protection has huge applicability and impact, there are no existing software patterns for developing a software system that explicitly deals with every aspect concerning it. The main problem that this research tries to address is ascertaining how to model software patterns for developing a stable application for consumer complaints and protection. A major goal of this work is to design and develop a generic application architecture that can be extended to numerous scenarios in related context. In addition, there are several other items addressed herein:.

- Different applications for the consumer complaints and protection concept have different perspectives and dimensions. How do we find out the core knowledge of the concept so that it is not specific to a single domain?
- After identifying the core knowledge, how do we make sure that this knowledge can be used in an abstract way so that different applications can be made independently?
- For a developer to analyze and design specific application scenarios, how do we supply the core knowledge of the concept so that they can extend that knowledge to generate the required application? (Fayad & Wu, 2002)

Solution

The SSM is the visual realization of software stability concept principles. The SSM modeling technique can be used to extract the core knowledge of the consumer complaint and protection domain (Fayad, Sanchez, Hegde, Basia, & Vakil, 2014; Mahdy & Fayad, 2002; Hamza, Mahdy, Fayad, & Cline, 2003; Fayad & Altman, 2001). By exposing this realized domain knowledge, a stable and generic core can be built to model any system in a related context (Davidow, 2003). In other words, by capturing the core knowledge about consumer complaints and protection, any related application can be modeled that shares this core knowledge. This enables any software designer or engineer to reuse this knowledge as many times as required (Howells & Weatherill, 1995).

According to the SSM, this core knowledge can be represented by Stable Analysis Patterns (SAPs) and Stable Design Patterns (SDPs). Analysis patterns are known for bringing down the price and time of the software development lifecycle. These patterns help designers to understand the core problem but do not show how to design an appropriate solution. On the other hand, SDPs allow designers to analyze the problem and generate the solution space by using the concepts of the SSM (Fayad, 2002a; Fayad, 2002b).

The SSM classifies all the classes of any system into three layers, namely Enduring Business Themes (EBTs), Business Objects (BOs), and Industrial Objects (IOs). EBT refers to the goals, aims, rationales, or purpose of the system. They are continuous, persisting, and never change. They are represented as SAPs. The BO always has a beginning and an end and represents the capabilities to achieve the goal (EBT) of the system. BOs are represented as SDPs.

In this thesis, we provide different SAPs and SDPs that can be combined to generate a stable architecture to be utilized across different domains and under different scenarios. Such an architecture exhibits unlimited applicability, as it is domainless because it makes use of the knowledge at an abstract level.

SSM Overview

The SSM allows us to build robust models built around a generic core. These models are stable and allow greater reuse across multiple applications (Davidow, 2003). The main objective of this thesis is to show how to model such stable and generic systems by using SSM concepts as the workhorses. But how can one use these ingredients of the SSM modeling technique into designing and developing stable products? Conceptually, SSM is divided into three levels:

- An EBT represents the goal of the application. It represents the objective of a system that is stable both internally and externally.
- BOs represent the capabilities or the application requirements. These are components that are adaptable internally but are externally stable.
- IOs represent the tangible components that vary with each problem scenario. In other words, IOs are the external interfaces for the whole system.

The hierarchies of these objects vary from the stable EBT level to the unstable IO level. The level of stability is influenced by the extent of tangibility at each level. EBT is always an intangible concept; BO is a partially tangible object, whereas an IO is a completely tangible entity.

Let us examine how these layers work in the SSM model. For SAPs, we start with an EBT as shown in Figure 1. Once we determine the EBT, we can then create many BOs based on a particular concept. However, we must first determine all the BOs that are needed for each one of the EBT scenarios.

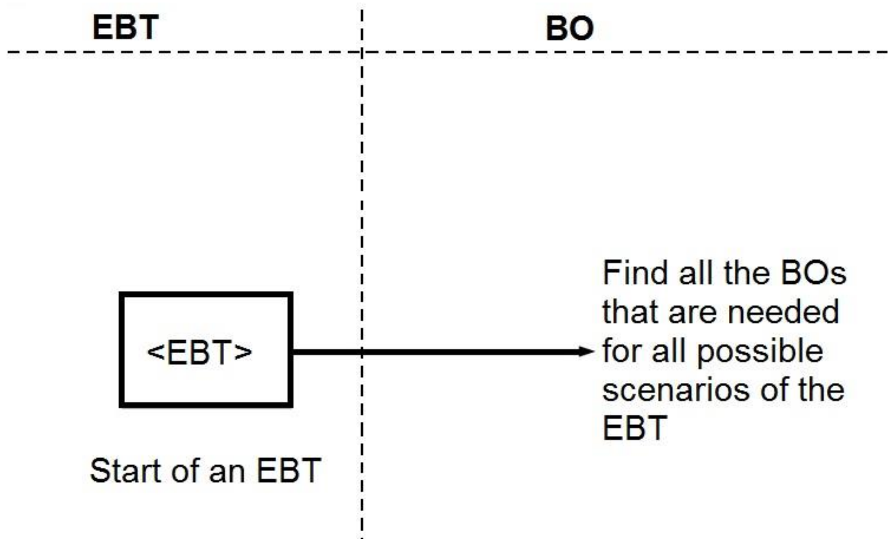

EBT
BO
<EBT>
Start of an EBT
Find all the BOs that are needed for all possible scenarios of the EBT

Figure 1. Level 1 for EBT requirements. EBT = enduring business theme. Adapted from "Extracting domain-specific and domain-independent patterns," by H. S. Hamza, H. A. Mahdy, M. E. Fayad, and M. Cline, 2003, *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications: ACM*, p. 310-311.

The EBTs remain constant for a given system. It is important, therefore, to choose only stable objects when creating them. The stability is also essential because the EBTs become the core of any software systems created from them. In fact, when we are creating the SDPs, we start with a BO built from the EBT as shown in Figure 2. Although BOs are also stable, the internal processes may require some of these to be modified slightly. Keep in mind, that the pattern is cyclic: Once we have all the BOs determined for a concept, they allow us to create a more accurate EBT. Figure 1 depicts approaching the project from an EBT standpoint while Figure 2 shows a project that began from BOs.

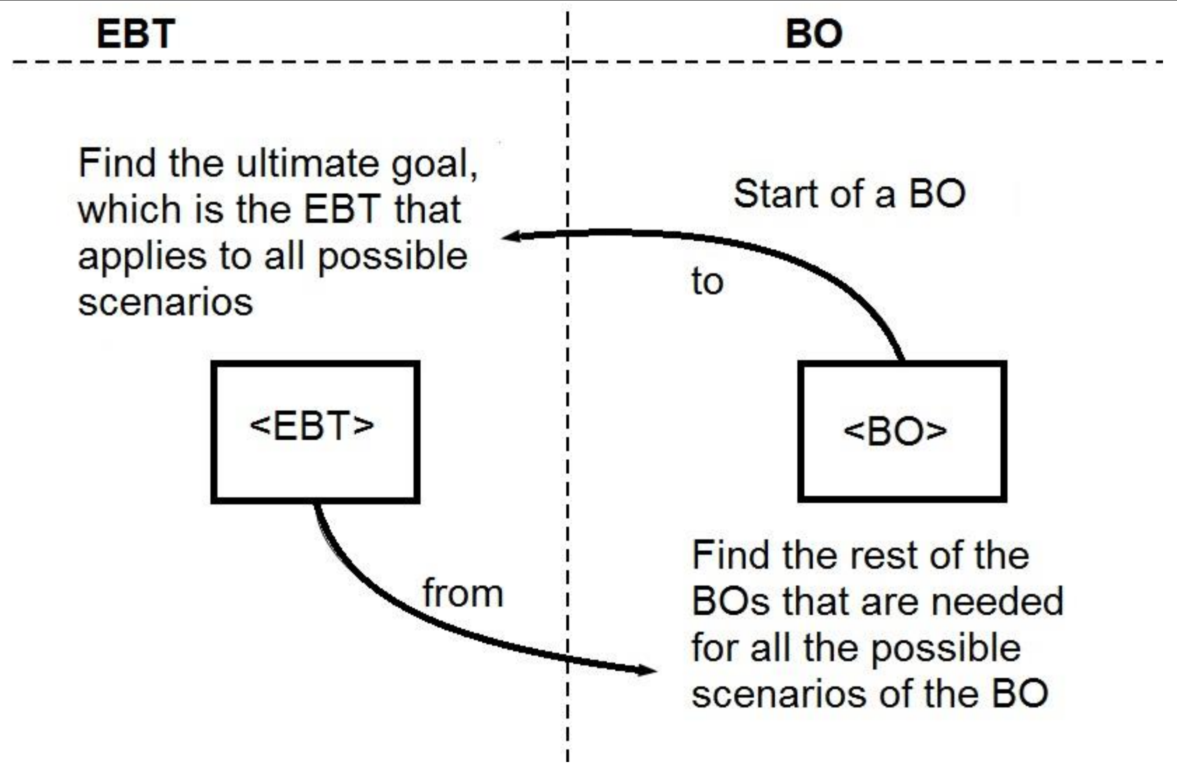

EBT
BO
Find the ultimate goal, which is the EBT that applies to all possible scenarios
Start of a BO
to
<EBT>
<BO>
from
Find the rest of the BOs that are needed for all the possible scenarios of the BO

Figure 2. Level 1 for BO requirements. BO = business object. Adapted from "Extracting domain-specific and domain-independent patterns," by H. S. Hamza, H. A. Mahdy, M. E. Fayad, and M. Cline, 2003, *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications: ACM, p.* 310-311.

Consumer Complaints and Protection Overview

Consumer complaints are complaints that are registered by consumers against sellers of any product or service they buy or use. A seller should not only sell a product or service to a consumer but also monitor customer complaints about the products and services they offer. If the complaint is not well addressed by the seller, then a consumer can take the issue to other organizations such as the Better Business Bureau (BBB) in the United States, which helps people find reputable businesses and monitors complaints about businesses.

Consumer protection shields consumers against malpractice, fraud, and poor business practices. A series of reports called Consumer Reports (CR) are based on different products and services that can be utilized by consumers. These reports provide consumers with a comprehensive analysis of the benefits and drawbacks of a specific product or service. Also, the CRs constantly watch out for misleading or unhelpful practices that stop a consumer from getting an satisfactory deal. The CRs' primary goal is to report the correct facts and information about products and services based on what experts and users claim.

Consumer protection laws, on the other hand, are prepared by assessing individual experiences with a product or service. However, most consumer protection laws take several months to navigate the system and become law. Before the laws are in place, CRs act as a trusted source for consumers before they make a purchase. CRs thereby make sure that the consumer voice is heard. It also empowers consumers to get the most out of their purchasing decisions by providing a mix of advocacy and advice. As a part of consumer protection, some common consumer rights formulated by rights activists include the right to safety, the right to information, the right to choice, the right to be heard, the right to redress, and the right to consumer education (see Figure 3).

A consumer complaint receives a more focused response from the business if it has more public exposure. Over the years, advancement in Internet technology and the subsequent rise of social media platforms has given consumers a new way to complain. Social media has proven to be the most effective way to file a complaint and to make it more visible. Being in the public domain and easily accessible, the complaints filed using the Internet help other consumers to make informed buying decisions.

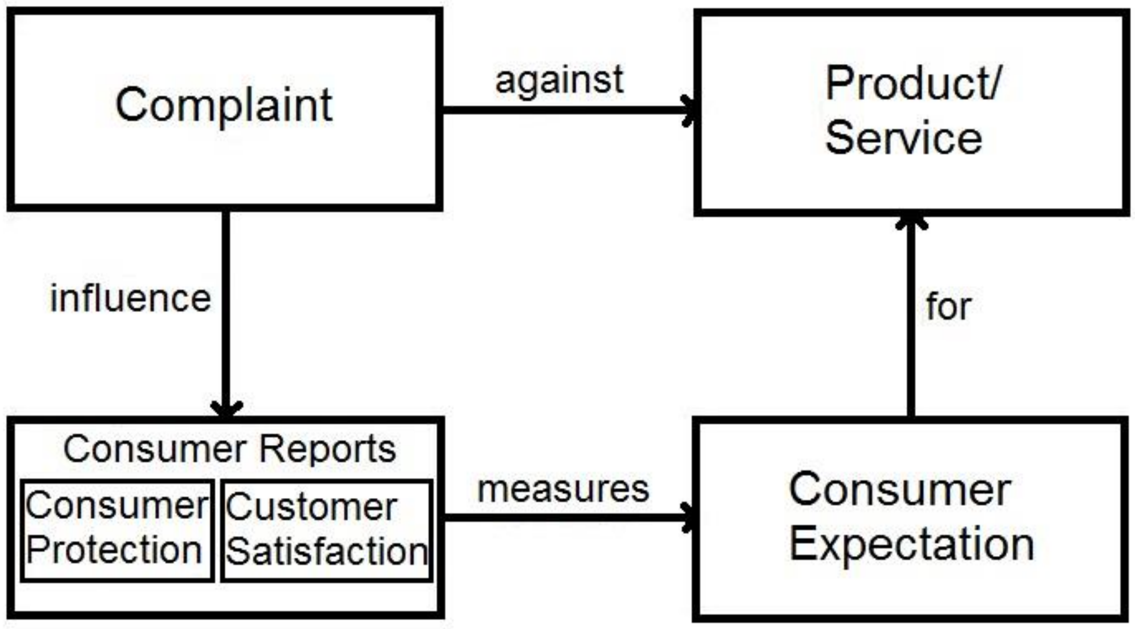


Figure 3. Block diagram of consumer complaints and protection showing how complaints affect consumer reports and products.

Research Methodology

When developing a system that deals with consumer complaints and protection, several key aspects or the core knowledge related to consumerism should be considered. This thesis is based on combining these essentials to form a generic core around which a stable system can be built. This core serves as an engine to design and develop unlimited applications of similar nature. The following methodology is used to create the aforementioned engine:

- Determine the goals and capabilities of a consumer complaints and protection program to identify the problem and the solution space. These components are categorized into analysis (EBT) and design (BO) patterns.
- The realized pattern structures were then used to identify the core knowledge and generate a knowledge map that is independent of any application-specific scenario.
- The core requirements collected through the knowledge map were analyzed and were collated to remove any redundancy.
- These requirements were used to design and develop a generic architecture that deals with all aspects of consumer complaints and protection. This architecture helps in building numerous applications in any related context.

Table 1 presents an illustration of different layers of the SSM modeling technique including the goals (EBT), the capability (BOs) layer, the software architecture layer, the development, the deployment, the quality factors, and the verification and validation layer. The research methodology followed is as listed below:

- Since the goal of this thesis was to provide the core knowledge of the consumer complaints and protection concept; it required reviewing published articles and research papers related to the concept as well as using logic, strategic, and brainstorming techniques.
- We analyzed data concerning methods of building a strong core that would allow us to add many applications to it.

- Research was also through periodic meetings with subject matter experts from organizations including Google and Yahoo, as well as other top professionals from around the world. These leading designers helped us understand the different layers required for the project.
- Several meetings with friends and fellow students were also conducted to design and develop patterns and knowledge maps for the goals of the system.

Table 1
The SSM Data Flow – Showing Different Layers of the SSM

Analysis/ Goals (EBT)	Design/ Capabilities (BO)	KM/ Software Arch Layer	Development	Deployment Quality Factors, V&V
• Identify the EBTs. • Encapsulate the core knowledge of the system. • Recognize the nontangible concepts. • Identify EBTs that are common to all applications. • Also called the analysis phase.	• Identify the BOs. • Find the classes that encapsulate the business logic. • Recognize the semitangible concepts. • Choose BOs that are common to all the applications of the domain. • Also called the design phase.	• Identify the core stable knowledge about the system. • Create a framework based on the identified EBTs and the BOs that is independent of any application logic. • The architectural layer lies in between the analysis and the design layers.	• Gather the application specific requirements. • Identify the IOs. • Implement the patterns developed in the analysis and design phases. • Implement the IOs specific to the application.	• Implement and integrate several patterns to build specific applications as per the requirements. • Quality factors. • Verify and validate the application. • Test the application.

Notes: KM = knowledge management; V & V = verification and validation; EBTs = enduring business themes; BOs = business objects; IOs = industrial objects.

Contributions

We provide detailed, medium and short templates of various stable analyses and design patterns identified as the core knowledge of the consumer complaints and protection concept. The contributions of this thesis includes the following:

- Two detailed SAP templates and two SDP templates.
- Three medium SAP templates and four SDP templates.
- Three short SAP templates and six SDP templates

The SAPs (Support, Compliance, Judgment, Need, Ownership, Fulfillment, Selling, and Promotion) and the SDPs (AnyAdvice, AnyClaim, AnyComplaint, AnyCommitment, AnyRate, AnyDeed, AnyModel, AnyView, AnyGuideline, AnyAppraisal, and AnyViolation) form the core knowledge of this concept. The list of knowledge maps that were generated using these patterns and presented in this thesis are:

- CRs Knowledge Map
- Services Knowledge Map
- Products Knowledge Map
- Service Provider Knowledge Map
- Complaint Knowledge Map
- Customer Satisfaction Knowledge Map

The significance of this thesis is in building a stable and reusable systems by designing unlimited applications over wide and cross-platform domains. We investigate the major problems faced when gathering the requirements and developing a system for the consumer complaints and protection domain. Developers may use the SAPs to analyze a problem and the SDPs to design an application in a related context.

Thesis Layout

The thesis is organized as follows: In Chapter 2, we provide a comparative study of different traditional patterns with the similar patterns built using SSM concepts. We also highlight the flaws in the existing patterns and present how stable models are better designed to overcome such flaws. In Chapter 3, we present detailed pattern documentation for four patterns, including both stable analysis and design patterns. In Chapter 4, we use a similar approach as in Chapter 3, but provide only average documentation for the patterns. In Chapter 5, we provide even shorter versions of templates for various patterns built using the SSM concepts. In Chapter 6, we offer an overview of future work and highlight the research issues of this thesis.

Summary

In this thesis, we propose a core knowledge of consumer complaints and protection. The patterns proposed form conceptual models, which help us to analyze and design applications based on consumer complaints and protection. The patterns developed in this thesis do not lose their generality, which enables them to be applied to similar applications across different scenarios. We recognize that all the objects in the proposed patterns have clear roles to play, and they are not dependent on the applications. The concepts of the SSM help us work toward achieving the stability goals of the system.

Chapter 2: Comparative Study

Traditional versus SSM Modeling Techniques

Software systems are logical yet complex creations of knowledge, which are designed to solve a problem or enhance ease of use for an activity or process. Complex software systems are usually developed by segregating them into smaller problems. For example, developers may examine solutions to smaller problems that then can be used for solving a bigger problem. In many scenarios, these subproblems are repetitive. In other words, there are some problems that were dealt with in earlier systems. These software patterns provide a way to document expert knowledge for creating solutions and apply it to recurring problems. They provide accessible knowledge that can be reused to build solutions for problems in similar contexts.

Another important dimension of software patterns is that the detection of such recurring classes or objects supports the process of reengineering and design recovery. This type of information is often represented using a pattern languages that describes and documents prior experiences in a cohesive and clear manner (Payne, 2006; Fayad & Singh, 2011). This chapter highlights the limitations of the existing pattern languages. It describes how a stable pattern language based on SSM concepts provides a better way to design robust software systems.

Software Patterns

In software engineering, a pattern is regarded as a solution to a recurring problem in any context. Put differently, a software pattern acts as a template to demonstrate how similar problems can be dealt with in different circumstances. A software pattern is expected to enrich the development process by putting forward some already tested development paradigms. But sufficient architectural knowledge is necessary while designing or creating concrete and robust software applications. Traditionally, software projects have shown high failure rates. Usually, the main reasons for software project failure are incomplete requirements and a lack of software adaptability. This is because the traditional methods of software modeling and design lack flexibility. However, the SSM technique relies on software modeling that is based on enduring core knowledge rather than relying on problem specific tangible artifacts.

One of the main problems with the traditional modeling approach is that it lacks the adaptability required to model stable software. Traditional models are built around tangible objects or classes that are specific to a problem scenario and are not reusable in nature. More often, such rigidity makes software systems prone to failure or prone to not adequately meeting the requirements of its users. Rigidity also tends to make applications less precise and difficult to execute because they often fail to represent the complete details of the problem (Steimann & Vollmer, 2006). The differences in implementation have made software applications more rigid and have restricted their broader adoption.

This chapter gives a comparison between some of the existing traditional software patterns and the stable design and analysis patterns. The SSM-based stable design and analysis patterns are an evolving field where new patterns emerge from time to time. This comparison reviews the advancements made in designing more stable and robust software using the SSM concepts.

We begin by illustrating the pitfalls of the existing modeling techniques and compare them to the SSM. We breifly discuss the main differences between the two modeling techniques. We also look at the different quality attributes each technique offers and conclude with a brief overview of how SSM is superior in most cases.

The Pitfalls of the Existing Modeling Techniques

While talking about software patterns, it is worth repeating that for any discipline, pattern knowledge forms the groundwork for repeated use of the domain information. This expert knowledge can be understood as a mirror of information gained through experiences over generations and can be used for future knowledge sharing (Hamza & Fayad, 2002). However, such critical information should be extracted and documented in a way that it can be easily understood and utilized. Collating complete and robust information through a pattern language is a major challenge. This is because such information can become useless if the core aspects of the domain are not covered completely. Traditional methods of modeling and designing software applications face this dilemma. These methods deal only with a specific problem and tend to ignore the core concepts of the problem domain, Thus, making resultant software prone to failure. The following are some major issues with the traditional modeling techniques:

Tangibility (lack of wide-applicability). UML incorporates various design paradigms and representations for modeling software applications. These models are meant to effectively communicate the design solutions to domain experts. The components of such models are represented using graphical notations that aid better understanding of the system components. However, traditional methods are focused on designing solutions for a specific problem statement. For this reason, these tangible components become problem specific in nature while limiting their wider applicability. Tangible classes and objects designed to resolve a specific problem prevents them from being reused or adapted to another solution even in a related context.

On the other hand, the SSM modeling technique is designed to focus on conceptual knowledge. Consequently, this helps in identifying the main aspects of the problem and the elements required to design a more widely applicable solution. The problem specific tangible components, called the IOs, implement application specific components. The purpose is to separate the core logic from the application logic. In this way, one can develop any number of applications that are based upon the core logic, and the application specific elements can be comfortably plugged in as per the application requirements.

Adaptability (rigidity to ever-changing requirements). With the rapid advancements in technology, modern software applications need to adapt quickly and efficiently. Using traditional modeling techniques, an application may not serve its complete purpose if the design has defects or is lacking regarding fulfilling the requirements. This is irrespective of how well an application is implemented using the design.

Considering the transient aspects related to a specific problem, the SSM provides goal-driven methodology that ties the fundamental core knowledge of the domainto the transient aspects of the problem. The outcome of this process is that it generates a set of knowledge in the form of interrelated patterns. The tangible components that are plugged into this knowledge and serve a specific purpose interact together as per the problem requirements, providing a more precise and efficient solution. This process can be literally repeated unlimited times to adapt to application needs. The flexibility when making changes and the easy adaptation to evolving needs gives the SSM a distinct edge.

Reusability (applicable in limited scenarios). The SSM provides multiple layers of abstraction from the core requirements of any domain through an enduring and stable collection of knowledge. This core knowledge helps establish a reusable base of artifacts known as EBTs and BOs that allows designers to build reusable software components. These components supply a means to reuse the design and analysis knowledge. They act as building blocks to the overall system architecture for designing a solution to any problem in a related context. This quality provides the software designers a medium for verifying and validating the application requirements each time and allows the designers to identify the best possible solution to the problem. It also provides stability to the architecture and design of the software system by detaching the domain core knowledge from unstable entities or those related to specific contexts. Therefore, through the use of interwoven knowledge in the form of patterns, developers can reproduce or redevelop complete solutions without reinventing the wheel.

Comparative Study

Any software modeled using the SSM concepts can be represented by the interconnected layers of EBTs, BOs, and IOs. The SSM concepts enable a complete understanding of the requirements and the architecture of any software system. While the EBTs represent the goals of the application or the problem space, BOs highlight the capabilities of the model or define the solution space. This complete model can be reused to design unlimited applications by attaching the IOs or the system classes as per the problem statement. For better understanding, this section provides an example from the existing pattern knowledge and presents an equivalent stable model developed using the SSM. The stable pattern is contrasted and compared with the traditional model using industry standard quality factors.

Software designed using the SSM modeling technique is more stable than the software designed using traditional methods. Consider the ordering and shipping a product as an analysis pattern (Figure 4). When observed closely, it can be seen that the pattern is more inclined toward the programmer's understanding rather than focusing on the core aspects of the problem domain. At first glance, the model appears to possess all the necessary classes required to design a shopping application. The customer has a profile and assumes the role of a registered user when shopping. He may add or delete items from his cart. The final shopping cart list is updated using the order class. Once the invoice is generated, the customer can pay using the methods defined in the payment class (by credit card, by invoice, or by direct debit).

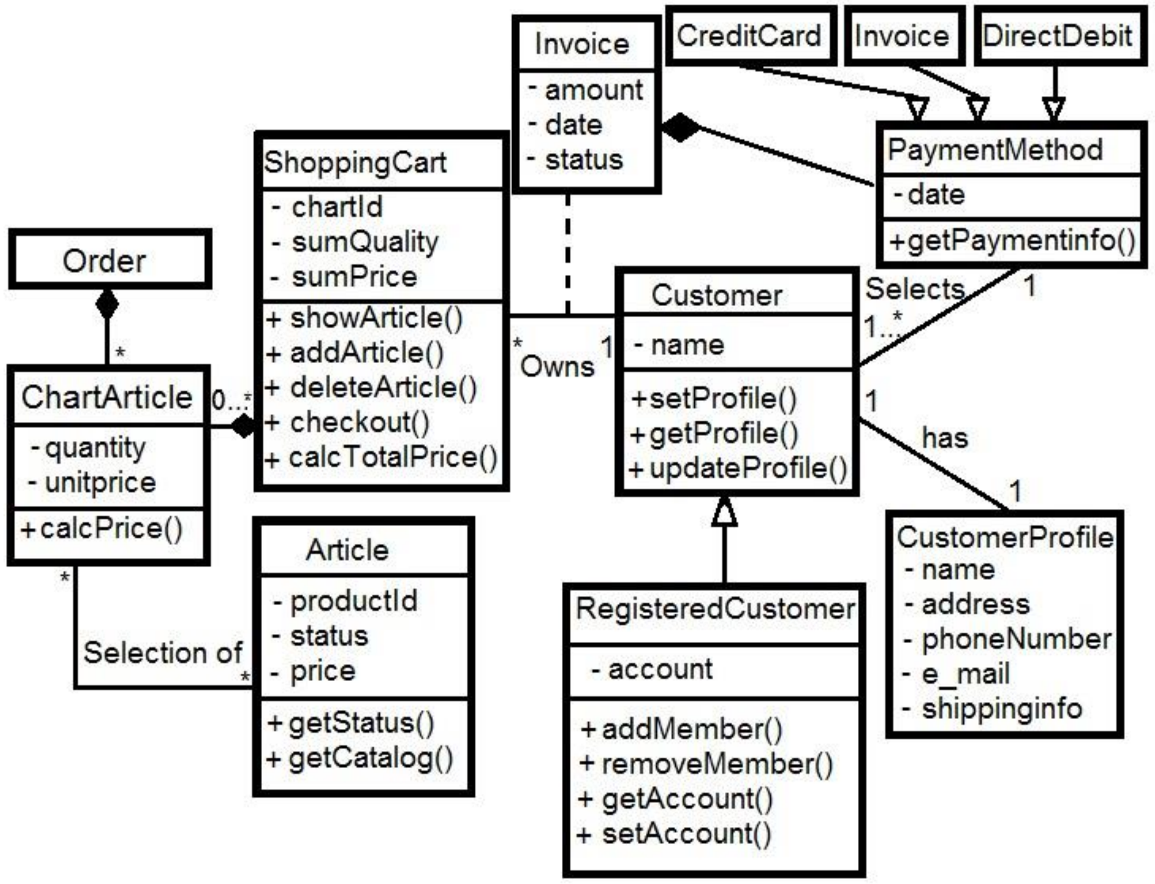


Figure 4. A traditional analysis pattern for ordering and shipping a product. Adapted from “Analysis Patterns for the Order and Shipment of a Product,” by E. B. Fernandez, X. Yuan, and S. Brey, 2000, *Proceedings from PLoP*.

Although the master class shopping process represents the application goal, it limits the design solution to this specific process. For example, if the same architecture were to be reused for ordering a free product or if the customer were not required to register for a purchase, the model would need substantial design changes. These changes are mostly influenced by the specific nature of the tangible system classes and by the dependency between the classes. Also, because of the limited abstraction from the specification details, the design is prone to develop inconsistencies between the requirements and the implementation.

On the other hand, the Selling SAP shown in Figure 5 focuses on the core concepts of the problem domain. This is intended to recognize the fundamental knowledge behind the complete selling and purchase process. The selling pattern does not limit itself to a single application, and it can be applied to any scenario just by attaching problem specific IOs to the pattern. All the core system classes belonging to the pattern are reusable and are easily adaptable to any internal change. A more detailed description of this pattern has been provided in Chapter 5.

Discussion

Developing a truly scalable and reusable application using traditional approaches is extremely difficult. This is because when a software developer tries to accommodate new requirements into an existing model, it causes a ripple effect on the entire application. On the other hand, the SSM, based on the foundations of core knowledge, allows adding, updating, or removing classes in sync with changing requirements. The SSM concepts make a model adaptable to such an extent that updating an application does not seriously affect the application structure. Table 2 illustrates the major differences between the two styles of the ordering and selling pattern.

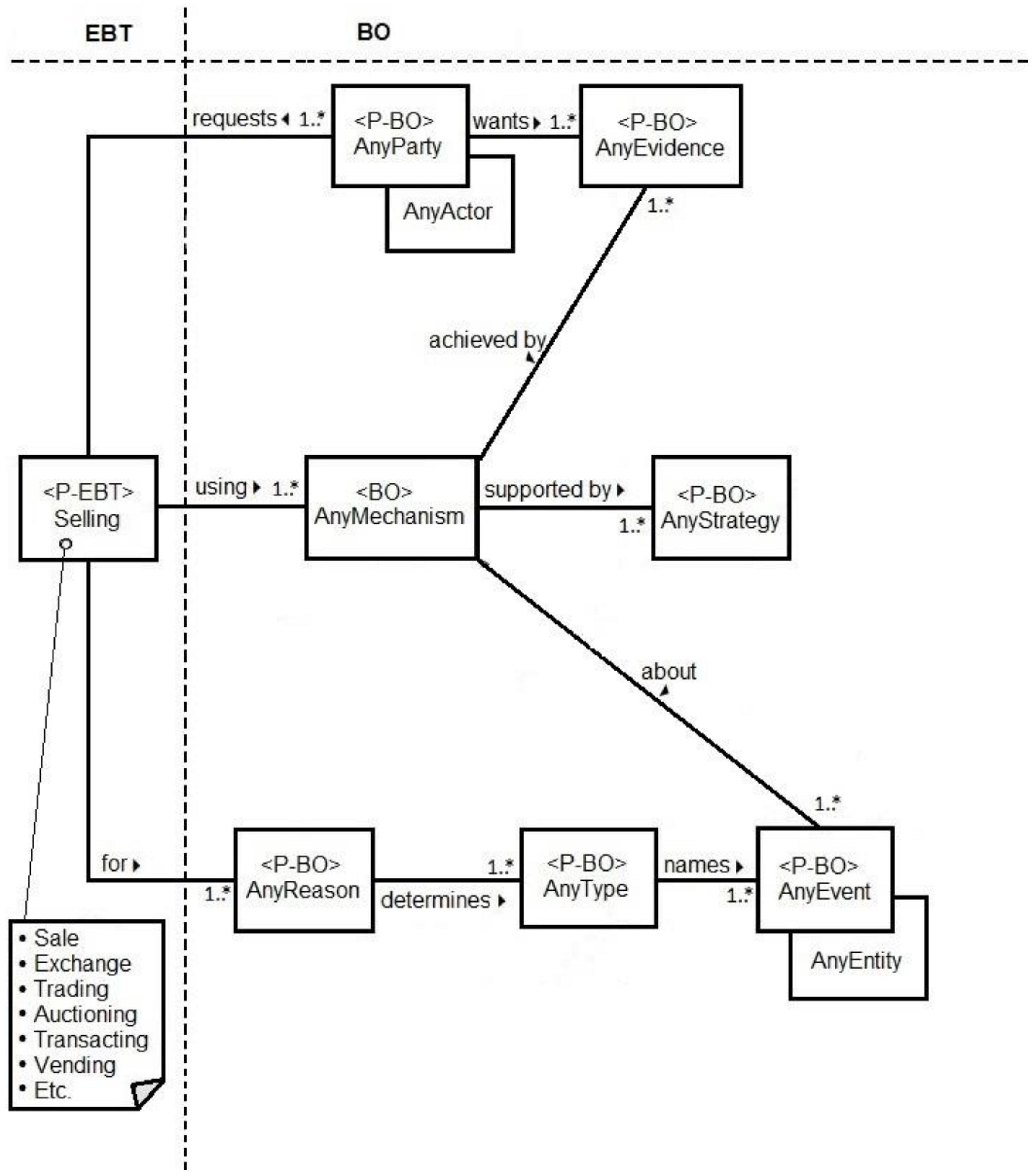


Figure 5. A stable analysis pattern for selling using the SSM. P-BO = pattern business object; P-EBT = pattern enduring business theme.

Table 2
Selling Comparison Between the Traditional Pattern and the SAP

Criteria	Description	Score	
		Traditional Pattern	SAP
Ease of use	The reusability and understanding of a pattern directly depends on its complexity.	5.5	9
Development costs	A reusable pattern that is not specific to one situation will incur fewer costs as it can be applied to other instances.	6	11.25
Flexibility	The level of generality to which a pattern structure can adapt to the changing requirements.	2	18
Dynamic analysis	The model's ability to make improvements to it during development.	11.5	17
Testability	The model's ability to be tested and validated before implementation.	6.5	9
Requirement fulfillment	The software pattern or model should be helpful and guide the development team.	8	13.5
Size of development	The total effort required to design and develop a software application regarding cost and time.	3.5	4
Quality of product	The ability of the final application to meet all the user needs and perform as per expectations.	2.5	4.5
	Total Score:	45.5	86.25

Note: SAP = stable analysis pattern.

Analysis

The traditional models are not flexible enough to accommodate new requirements without a considerable effort. But the SSMs allow developers to incorporate changes in the system without actually impacting other parts of the system. Such models can be scaled either up or down depending on the change in requirements (Fayad & Jindal, 2015). Qualitatively, there are also several differences between the traditional model and the SSM.

Abstraction (level of detail). Any software designed using the traditional method is either too abstract or too detailed. Abstract models are usually easy to understand but in traditional models it is a tedious process to convert them into more detailed structures. Furthermore, such a conversion depends on the expertise and knowledge of the software designer. On the other hand, too much detail makes a design model much more complex and difficult to understand. The SSM overcomes these problems by concentrating fundamental knowledge into a stable core and by only exposing the variant aspects of the application to the designer. This helps with segregating the information and with enabling the software designers to focus on dealing with the application-specific requirements.

Applicability. Software applications modeled using traditional methods possess tangible classes. This makes them problem specific and inflexible to changes so that they cannot be adapted to work for any other problem even in a related context. On the other hand, the SSM methodology is based on the core knowledge of a domain. It gives the application design much-needed stability regarding solution requirements. This aspect of the SSM based models allows them to be easily adaptable and work across various scenarios. Refactoring and maintenance of the application code are easy because all the requirements are satisfied during the initial model building using the EBTs and BOs. Thereafter, adding, updating, or removing further requirements becomes very simple.

The impacts and advantages. Traditional models usually do not cover all the application requirements, which forces them to keep updating the system whenever necessary. Any SSM based models can be used to model stable, reusable, and enduring software solutions. As the SSM possess fewer dependencies between the classes, they remain stable even during a change in application, design, or architecture. On the other hand, core concept-based knowledge helps build system classes that can be reused as many times as required to design similar solutions. Finally, since the SSM models are devoid of tangible classes and are developed as goal-based systems, they can be used for designing long-term solutions.

Conclusion

The aim of this chapter was to compare the existing modeling techniques with an SSM based technique. Usually, the application requirements evolve over time. Therefore, it becomes a tedious process to update the existing model. In the case of complex systems, such a process can make the job of a software designer even more arduous because any new requirement requires the designer to refactor the code. Furthermore, as the traditional method complexity increases, it becomes increasingly less understandable. The SSM provides a way to overcome these issues and introduces an enhanced way of modeling a stable and reusable software system. It is not only more efficient but also builds better software systems.

Chapter 3: Detailed Documentation of Stable Analysis and Design Patterns

In this chapter, we provide detailed documentation for three patterns related to consumer complaints and protection, namely

- the support SAP,
- the compliance SAP, and
- the AnyAdvice SDP.

Support SAP

Pattern name. Support is an EBT. It means to provide assistance or to help someone or something (“Support,” 2016). The term signifies AnyType of aid, benefit, cooperation, sponsorship, cooperation, or advice. Generality is the primary motivation for choosing the term support in this scenario is because it is appropriate for all support situations. It takes distinctive values, but each has the same importance and value. A systematic process of the software stability concept (SSM) was used to develop the support concept into a more relevant and useful pattern.

Context. Support, in general, has applications in various domains such as psychology, finance, chemistry, mathematics, military, politics, and sales. Some of the commonly known types of support are life support in hospitals, technical support from businesses, peer support in educational institutions, income support from charitable and government agencies, catalyst support in chemistry, and combat support during wars. There are two support situations we will highlight.

Perhaps the first thing most people think of when hearing the word support is technical support. Consider a scenario in which a client such as San Jose State University

(SJSU, AnyParty) buys MS Visio (AnyEntity) from Microsoft (AnyParty), but the SJSU is unable to install (**AnyReason**) MS Visio on their computers. SJSU then seeks technical assistance (support) from Microsoft because the support was promised for MS Visio for free for the first 30 days (AnyConstraint) after purchase. Microsoft will have a technician (AnyParty) troubleshoot the MS Visio related problems by the way of their remote computer repair (AnyMechanism) system. The technician, with prior expertise (AnyLevel) and troubleshooting skills (AnyResource), would help the client install MS Visio.

A remote computer repair mechanism is an advanced system used by maintenance technicians to gain access to a client's desktop with their permission through the Internet. The technician uses the access to conduct maintenance and repair remotely on the client's machine (AnyType). The reason for the installation failure (AnyReason) was due to the lack of necessary technical skills. In the process of maintenance and repair, the client will be able to successfully troubleshoot (AnyImpact) MS Visio installation problems with the technician's support.

Another important area of support is moral support. Moral support has a psychological aspect (AnyMechanism), and it is not related to material help. An employee (AnyParty) wants a raise (AnyImpact) in his or her salary (AnyEntity). A coworker (AnyParty) encourages the employee and helps him or her address any potential disagreement with the employer (AnyParty). With the colleague's moral support, the employee approaches the manager and explains the hard work they have done (AnyReason) and the serious efforts made to reduce the cost of production

(AnyEvent). The department manager has a policy of raising staff salary only when they score a minimum of eight or above (AnyConstraint) out of ten points on the annual review. The employee had not done this successfully. The coworker helps and encourages the employee, enhancing the employee's confidence level (AnyLevel) by asking him or her to dedicate more time, focus on their daily work, and to develop rational thinking (AnyResource), which will eventually help the employee score better on the yearly review ratings. Then, the employee not only gets a score of above eight in the annual review rating, but also becomes successful in getting a raise. Hence, the emotional (AnyType) help provided by a coworker becomes key in assisting the employee to receive a raise.

Problem. The main problem with addressing the flaws of current consumer protection systems is to create and derive the core knowledge of the support concept so that the fundamental constituents are covered in complete detail (Fayad & Hamza, 2004). The main question to ask here is: "How can one build a model that will tolerate a fair level of abstraction?" (Fayad, Hamza, and Stanton, 2004). Creating such a model is neither easy nor flexible. Ideally, it would be a single, generic model that could be used to manage all existing problems across all domains.

Functional requirements.

- Support is the EBT for this pattern and the ultimate goal. For example, customer support in businesses means the vendor offers assistance to the clients. Support has common attributes such as degree, means, and category. The degree of support relates to the extent that the support is provided. The means of support is the way that the intended support is provided. When support is provided, it is important to know the category of support extended: Is it financial, educational, psychological, or material? Support needs operations like assist(), including maintain(), sponsor(), defend(), and encourage().

- AnyParty offers support. Support is either sought or offered by and individual, an organization, a country, or a political party. AnyParty will generally have a name, contact information, and a phone number. These modifiers are important because a name will help to identify a particular party easily, and the contact information provides a method for contacting the party. A support employee of an organization participates() in company projects or playRoles(), by collectingData() and by interact()ing with other employees and customers.
- Some support is not provided by a party but rather by AnyActor.   For example, Microsoft Word (AnyActor) is used for the insertion of text, images, and other input. AnyActor will have a name, unique ID, role, and category. The support actors will need to explore() in order to find support providers, request()support, and receive() it accordingly.
- Support is usually triggered by a mechanism and is a means of generating required results. AnyMechanism is machinery that helps to get the desired result. A mechanism will have a context, id, name, status, and application. In order to achieve the desired results, AnyMechanism would execute() actions when activated(). The status() of the mechanism indicates whether the mechanism is, for example, paused(), attached(), or detached().

- Support has AnyLevel degree of intensity with which it is looked for or provided. Depending on the level of support, there will be an impact on the parties involved. In the case where someone wishes to provide moral support to another person, then the level of support would be superior in a personal meeting as compared to the support extended through phone calls or emails. AnyLevel has an id, type, unit, and context. AnyLevel decides() the amount of support and can be classified() into types of levels such as old() and new(). AnyLevel of support may also change().
- Some conditions or requirements influence AnyReason. Every support has a reason, and reasons are of different types. AnyReason has attributes such as description, condition, and statement. AnyReason often has a description. Whenever one gives a reason, the details of the reason should be mentioned for better understanding. AnyReason should help figureOut() speculation and decide(). AnyReasons help to prove() explanations for actions.
- There could be different AnyTypes of support. A few examples are military support, moral support, technical support, child support, and income support. AnyType has a class, name, ID, property, and subtype. AnyType allows support to classify() and change() based on the sorting(). One example would be cars, which can be classified() into several subTypes() based on what they operateOn(), and these types change() after a period of time.
- Support is provided to and given by AnyEntity. AnyEntity is any product or service that exists. For example, pillars or columns provide support for the entire

structure of a building. Here, the building is AnyEntity. AnyEntity has a name, id, and type. AnyEntity could be a machine expected to performFunction(). Two small machines that are part of a bigger machine will have a relationship() among them. AnyEntity will also have a status() depending on its operation and will be classified as new() or old().

- Support is provided for AnyEvent, too. In a marriage ceremony, family members attend the ceremony to support the marriage. In a game of soccer, fans watch the match in the stadium to support their favorite team. AnyEvent has a name, outcome, and type. AnyEvent occurs() during a particular timeFrame() and is performedBy() someone or something to produce a result().
- AnyResource is the capability or source with which support is provided. It can be a system, a talent, some capital, or even an appliance. AnyResource has a name, a supply, and a system. AnyResource connects() different entities and events to each other. It allows AnyMechanism access() to support. AnyResource can be extended() irrespective of it being a material or nonmaterial object.
- AnyImpact is an effect caused by a support action. When using AnyMechanism provides support, the result of that support is AnyImpact. It could be an influence, an impression, or even a burden. AnyImpact has a style, a strength, a force, and a level. When there is AnyImpact, it is measured() by its effects() and its significance().
- AnyConstraint can be a force or restriction bestowed upon an actor or a party. These constraints are specific to the support being provided. AnyConstraint has

an id, a type, a limitation, and a state. AnyConstraint is needed(), and it can be classified(), found(), created(), or deleted().

Non-functional requirements.

- Support has to be acceptable. For example, it is not acceptable for a grocery store to sell rotten food. According to Oestreicher (2007), many people are pushing to develop robots that are highly cognitive. These robots can support human needs. However, in some countries, such robots are not socially acceptable, and so they would not be able to support humans that live in those countries. In another example, numerous schools and universities create user policies that involve usage of campus Internet and e-mail by staff as well as students (Olds, 2000). Frequently, the university will not allow their Internet and e-mail services to be used for the creation or distribution of offensive material. It is important for a pattern, whenever it is applied or provided, to be acceptable.
- Every instance of support is intended to be helpful. Helpfulness might include some form of encouragement, financial aid, or even comfort. For example, lending money to a friend helps them in a time of need. In another scenario, people are supported by winning votes or receiving comments (Liu and Karahana, 2015). In fact, some comments are used to determine whether the information presented on a website is helpful or not. Thus, the user who provides comments actually supports other users of a website. However, consider a man who is running a race. If someone trips the man, that person is not being helpful. The person is not supporting the runner.

- Support needs to be effective. For example, military support that helps the army win a battle is effective. If the support comes too late or is not powerful enough, it is not effective support. Similarly, technical support specialists should have good communication skills so that they can understand clients' needs and communicate solutions clearly and coherently (Cerri, 2000). If the technical support specialist misunderstands the client and presents the wrong solution or if the client misunderstands the specialist and the problem is not resolved, the support was ineffective. Thus, it is important for support to be effective.
- Any support provided should also be sufficient. Anyone who seeks support expects the support obtained to be sufficient. For example, consider life support, which is used in an emergency to support life. If the life support does not provide enough oxygen to the person who needs it, the person could die. Similarly, if a person supports his or her child with money to pay college fees, but only gives half of what is needed, the child will not be able to attend classes and obtain his or her degree. In another situation, the death of a loved may cause great distress. Without sufficient support from friends and other family members, the distress may grow to the point where it eventually affects work and daily schedules. It may continue to escalate until it affects the person's health. Thus, it is important for support to be sufficient.

Challenges and constraints. Support has both challenges and constraints that restrict its ability to function properly. Things such as a lack of resources, a lack of mechanisms, lack of relevant resources, lack of relevant entities, unexpected impacts, unknown entities, and out-of-date entities can all challenge support and constrain it. A bank (AnyParty) provides technical support to its customers (AnyParty) through e-mail (AnyMechanism). The bank uses resources earned from customer loans (AnyResource) to provide this service free of cost. However, the e-mail support is insufficient and slow especially as the number of customers increases over time. The bank should extend its technical support to live chat and telephone. However, this would increase the number of mechanisms for providing support. The amount of resources (money) needed may not be sufficient to provide these new resources. There should be a balance between the mechanisms for support and the money required to fund those mechanisms. It is important for the bank to find this balance, possibly by increasing loan interest rates to fund the resources better.

When an earthquake occurred in Nepal (AnyParty), countries all over the world extended their support in various ways. Pakistan (AnyParty) was one of several countries that sent ready-to-eat (AnyType) beef (AnyEntity) as a resource (AnyResource) to Nepal. However, cows are considered sacred to the majority of Nepalese people, and beef is not eaten by them (AnyConstraint). Therefore, the support was not relevant to their cultural needs. Entities providing resources as support should ensure the resources are relevant. In this case, cultural cuisine should have been better researched and vegetarian or nonbeef meals should have been sent.

In a war, it is quite common for ground forces (AnyParty) to request air support (Support) to help destroy enemies (AnyParty) on the ground. Pilots (AnyParty) and ground forces coordinate with each other to make an air strike (AnyMechanism) on the enemy. If there were improper communication and coordination (AnyConstraint) between pilots and ground forces, an air strike may affect friendly forces and cause deaths or injuries (AnyImpact). To resolve this issue, there has to be proper coordination between the air and ground forces to avoid unexpected and problematic results or consequences. In addition, a series of checks should be in place that prevents accidents from happening such as visual confirmation of targets.

Another example would be that of a mobile service store (AnyParty) that provides technical and product support for Samsung phones (AnyEntity). The Samsung phone is AnyEntity, and whenever it does not work as expected the store should provide support. The store receives a hundred phone requests per day (AnyConstraint), and the service center easily repairs (AnyImpact) all of them. However, if a virus hits the phones and the number of needed repairs increases to thousands, and then the same service center with the same number of staff would not be able to meet the new support demands. It would take an inordinately longer time to service all the new requests.   The entities for which support is being provided should be definite or pre-defined in accordance with the number of resources available, the time required, and the amount of money involved.

The Microsoft Corporation (AnyParty) provides technical support to its clients (AnyParty) for their operating systems, such as Windows 8 and Windows 8.1 (AnyEntity). However, as of April of 2014, support for Windows XP was terminated. The standard support duration was ten years (AnyConstraint) for Microsoft products, which prevented it from continuing support for Windows XP. Microsoft suggested that its customers upgrade their operating systems to Windows 7 or Windows 8 to continue support. Entities need to be upgraded to prevent overburdening support. Users should be informed about the guidelines and constraints of technical support in order to receive continuous support.

See Table 3 for a brief comparison of these constraints and challenges.

Table 3
The Challenges and Constraints of Sample Support Scenarios

Problem	Problem	Constraint	Solution
Technical Support at a Bank	A bank using e-mail support does not respond quick enough. Support is funded from interest on loans.	Balance between resources and mechanisms.	The bank needs to expand support to phone and live chat. The bank may have to raise rates to fund this.
Earthquake in Nepal	Countries sent beef to Nepal as aid, but the Nepalese people consider cows sacred and do not eat them.	Donating relevant resources.	Countries needed to provide nonbeef or vegetarian aid meals after the earthquake.
Air Support During a War	Poor communication between ground and air forces may cause damage to friendly units.	Unexpected impact.	Communication must be improved between air and ground forces. Checks should be in place to prevent accidents.
Mobile phone support	A virus causes a service center capable of servicing 100 phones per day to service 1000 phones per day.	Indefinite entities require support.	The number of entities to support should be clearly defined and definite. Resources should be distributed according to needs.
Support for Windows XP	Microsoft only continues support for operating systems for up to ten years. Windows XP is no longer supported.	Support limitations need to be clearly defined.	Microsoft recommends upgrading to a newer operating system when old operating systems are no longer supported.

Other support constraints include

- AnyActor or AnyParty provides support for AnyReason, and AnyMechanism triggers it;
- AnyActor or AnyParty defines one or more constraints;
- AnyConstraint influences AnyMechanism;
- support triggers AnyMechanism, which AnyReason determine and AnyConstraint influences;
- AnyLevel depends on AnyResource;
- AnyReason names AnyType, and AnyMechanism determines it;
- AnyType determines AnyEntity or AnyEvent, and AnyReason names it;
- AnyType determines AnyEntity or AnyEvent.

Solution. We hereby provide a working solution by suggesting a support SAP that uses the SSM concept. The given solution can help design and build numerous applications on a series of cross-platform domains.

Pattern Structure. The relationship between the EBTs and BOs within the support SAP is shown in Figure 6. The figure gives the class diagram for the support patternOn it; there is a brief introduction to each class and its role.

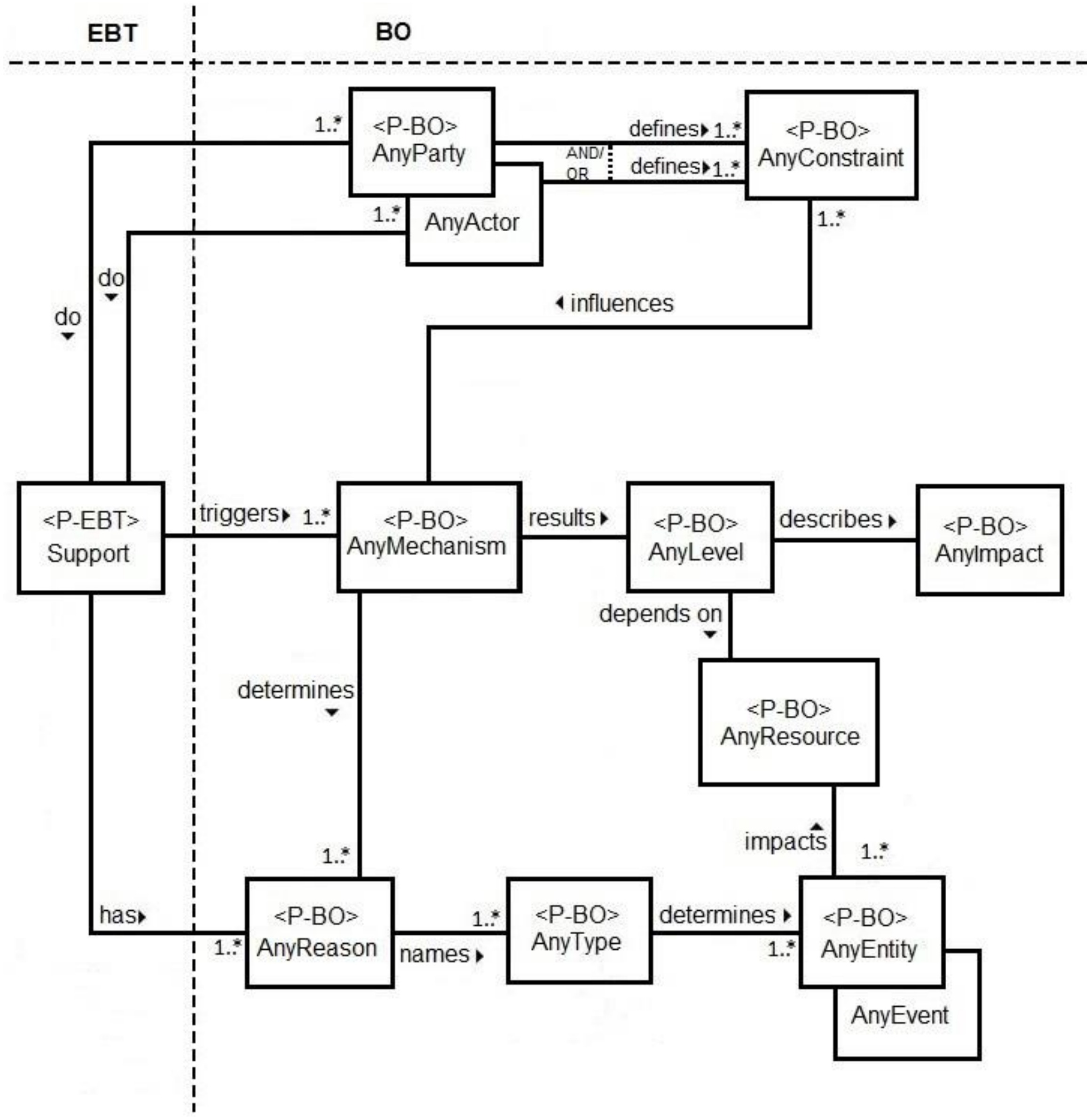


Figure 6. A class diagram of the support stable analysis pattern. P-BO = pattern business object; P-EBT = pattern enduring business theme.

Pattern Participants. The participants within the pattern are the EBT and BOs. We describe them as classes and patterns. Classes represent the support process as it is in its original state. This forms the core of SAP presented here. Various attributes and operations control the support process. Association classes, constraints, interfaces, tagged values, and notes are included in the class diagram. The class diagram also shows how we to connect the classes to each other to build stable applications with it.

Patterns contain the following elements:

- AnyParty represents both the party that provides and the party that seeks support. It models all of the parties in the support process. AnyParty could be a person, organization, country, or political party.
- AnyActor represents the actor that provides or that seeks support. It models all of the actors participating in the support process. AnyActor could be software, hardware, a person, or a creature.
- AnyMechanism represents the mechanism that is used to perform the support operation.
- AnyLevel represents the required level of support as recognized by an actor or a party. An example of this would be calling the level inferior that uses e-mail support. This is an inferior type of support because it may take few days to receive help desk responses. It is the level or degree of support that is rated here. The highest level of support would be an on-site technician.
- AnyReason represents the reason support is needed or provided. It could be a fact, a situation, an explanation, a rationale, or a condition.

- AnyType represents the different types of support that can be provided or that are needed by an actor or a party. A few of the support types are moral support, military support, and financial support.
- AnyEntity represents the entity that needs to be supported. It can be any object such as a phone or laptop.
- AnyEvent represents the event that needs to be supported such as protests, marriages, and emergencies.
- AnyResource represents the resource through which the support is extended. For instance, a support technician will use his or her expertise and problem-solving skills to provide support for equipment.
- AnyImpact represents the effect that is the visible result of the support process. Support usually causes an impact or an influence upon the actors or parties as well as the entity or event.
- AnyConstraint represents the limitations or constraints of support processes, such as any external factors that can hinder ongoing support operations.

CRC Cards. CRC cards summarize the responsibility and collaboration of each participant. These cards can easily be organized and ordered throughout the process. Each participant has only one well-defined responsibility on its CRC card. Table 4 defines the information needed to fill in CRC cards for each EBT and each BO central to the support SAP.

Table 4
CRC Card Information for the Support SAP

Participant	Duty	Attributes	Collaboration	
			Client	Server
Support (EBT)	Provide assistance	domain, name, context, supporterInfo, feedback, phoneNumber, satisfaction	AnyParty**,** AnyActor, AnyMechanism, AnyReason	provideAssistance(), calculateSupport(), abideBy()
AnyActor (BO)	Seeking support	age, name, id, gender, email, phoneNumber	AnyConstraint. Support	receiveAid(), addDuty(), approve()
AnyParty (BO)	Seeking support	service, name, address, phoneNumber, email	AnyConstraint, Support	pursueHelp(), giveEncouragement(), stopAssistance()
AnyMechanism (BO)	Generate results	id, name, status, application, description, context	AnyConstraint, Support. AnyLevel, AnyReason	execute(), activate(), deactivate(). pause()
AnyConstruct (BO)	Enfoce limitation	restriction, type, severity, threshold, description	AnyParty, AnyActor, AnyMechanism	exerciseRestriction(), ontrol(), obstruct
AnyLevel (BO)	Define scale	id, name, type, amount, size, degree, volume	AnyMechanism, AnyImpact, AnyResource	scale(), measure(), align, describe()
AnyImpact (BO)	Effect	id, name, level, xonsequence, force, power	AnyLevel	effect(), weight(), sustain(), influence(), push()
AnyReason (BO)	Specify details	id, description, name, proof, limit, case	AnyMechanism, Support, AnyType	analyze(), explain(), defend()

AnyResource (BO)	Supply support material	id, resourceName, talent, category, means, natureOfResource	AnyEntity, AnyEvent, AnyLevel	reserve(), handle(), increaseAbility(), supplyMoney()
AnyType (BO)	Designate category	id, property, interfaceList, methodList, subtype, typeName	AnyReason, AnyEntity, AnyEvent	classify(), operateOn(), attach(), detach()
AnyEntity (BO)	Exist independ-antly	entityName, id, type, position, state, status, isAlive	AnyType, AnyResource	performFunction(), relationship(), new()
AnyEvent (BO)	Exist independ-antly	name, id, type, occasion, contract, contender	AnyType, AnyResource	compete(), celebrationFor(), rank()

Consequences. The consequences of creating an SAP could be far-reaching regarding accommodation and materials. The obligated BOs are inappropriately coupled, which forces them to be hooked to countless IOs according to the need. When a pattern is distinguished from the support SAP, the main problem that remains is to distinguish the IOs and unite them with suitable hooks. The benefits of creating a stable pattern are all the following:

- The pattern is scalable because the hooks and their detached coupling gives the pattern an incredible stability.
- Extensibility is ensured because of the expansive number of IOs to BOs.

- This stable pattern is exceptionally interoperable because it can be identified with other stable patterns.

Table 5
Applicability of the Support EBT Across Several Disciplines

BO	Life Support	Moral Support	Social Support	Child Support	Technical Support
AnyParty	Doctor, patient	Son, father	Student, teacher	Parents, child	Microsoft, SJSU
AnyMechanism	CPR	Presence	Concern	Periodic payment	Remote computer repair
AnyReason	Cardiac Arrest	To encourage	Low grades	Unable to get along	Unable to install
AnyType	Physical	Emotional	Emotional	Financial	Local outsourced
AnyEntity	Heart				MS Visio
AnyEvent		Baseball match	Exams	Divorce	
AnyResource	Blood Flow	Cheer	Study skills	Money	Computer skills
AnyLevel	Expert	National	Personal	Income level	Expert
AnyImpact	Keep alive	Perform	Better grades	Care	Troubleshoot
AnyConstraint	DNR order	Noise	Number of students	Neglect	30 days

Notes: CPR = Cardiopulmonary Resuscitation; DNR = Do not Resuscitate.

- The pattern is flexible and could relate to many uses, since the base pattern is usually open to outside associations. There are many ways to apply this stable pattern to other problems. Consider the examples in Table 5 and how they relate to the pattern.

Case Study 1: Support for Life

In a situation where a patient (AnyParty) experiences a cardiac arrest (AnyReason), an EMT (AnyParty) might have to use CPR (cardiopulmonary resuscitation; AnyMechanism) to provide emergency care that keeps the patient alive (AnyImpact). CPR is a technique that is used to save lives in emergency situations. When the patient's (AnyParty) heart stops beating, CPR can continue the process for him or her by aiding blood flow (AnyResource) through the body. The support CPR provides is physical (AnyType). Some common emergencies that require CPR are heart attack, pulmonary arrest, or electrical shock (AnyReason). CPR certification (AnyMechanism) may be obtained on several levels (AnyLevel), from basic to expert, and it can be used to restore the functioning of the heart (AnyEntity) during a cardiac arrest, unless the patient has a Do Not Resuscitate (DNR) order in place (AnyContsraint). If a patient's medical records indicate that patient has refused to be resuscitated, the healthcare providers must honor this (AnyConstraint). Figure 7 provides the class diagram of this case study.

Use case and description. With each class there are certain questions to consider as we move through the steps of the design.

- Support can provideAssistance() to AnyParty(). ((What assistance is provided? Who provides assistance? How is the assistance provided?))
- Support can provideAssistance() to the patient.
- AnyParty() can diagnose() victim and requestVentialtor(). AnyParty() also may stopCPR() (What does AnyParty() diagnose? From whom does AnyParty() requestVentilator()? Why does AnyParty() stop CPR?)
- HealthCareProvider diagnoses() patient.
- Patient can request() CPR from the HealthCareProvider.

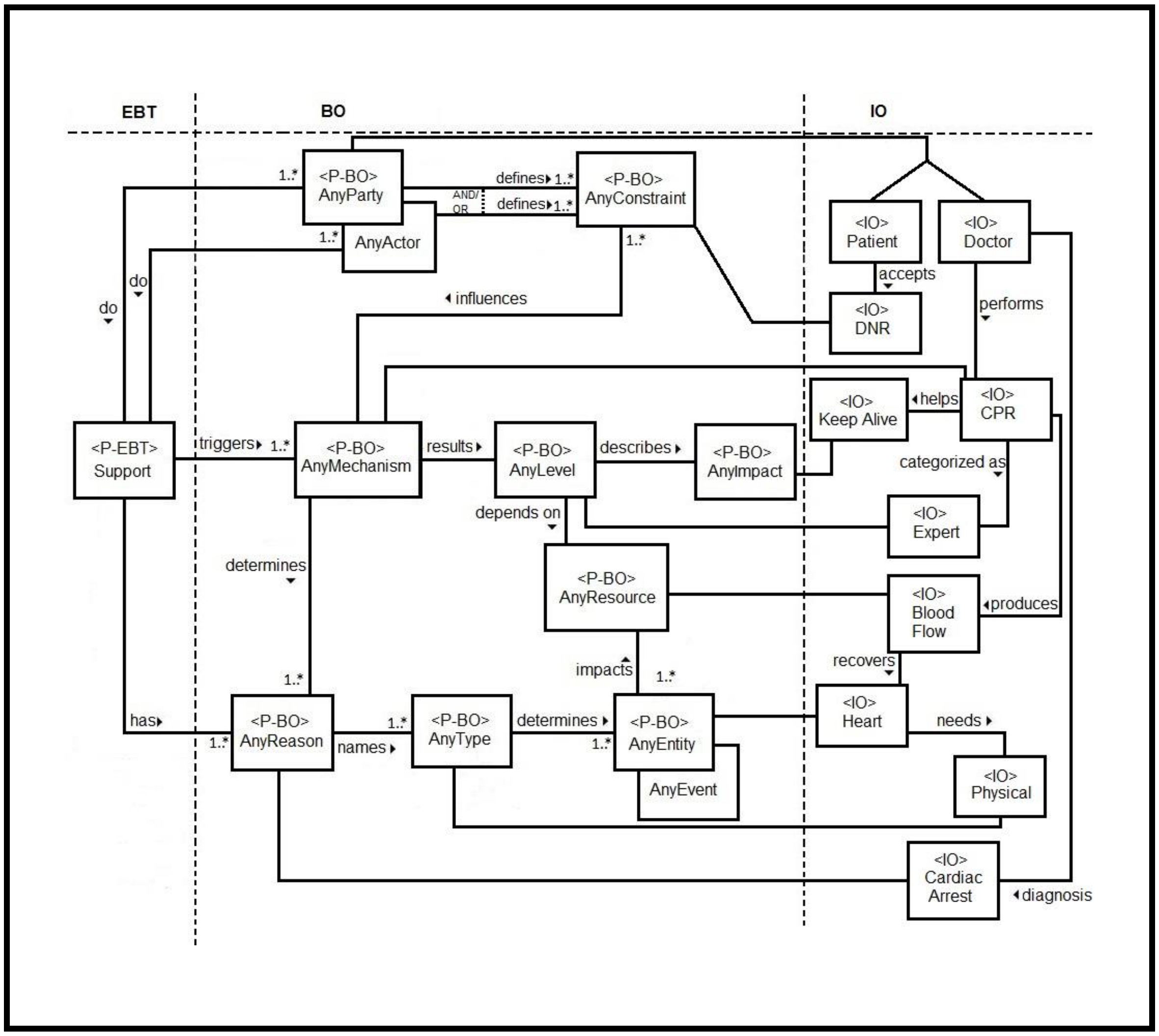


Figure 7. The class diagram for life support. showing the EBT, BOs, and IOs.

- HealthCareProvider uses AnyMechanism() to restoreCirculation() in AnyEntity(). (What circulation is restored by AnyMechanism()? How is circulation restored by AnyMechanism()?) For example, using CPR can restoreCirculation() in Heart.
- AnyReason() can carrySymptom() to AnyParty(). (What symptoms does AnyReason() carry? To whom are the symptoms carried?)
- CardiacArrest can carrySymptom() to HealthCareProvider().
- AnyType() can regulateRespiration() of AnyEntity(). (How does AnyType() regulate the respiration? Whose respiration is regulated by AnyType()?)
- Physical actions can regulateRespiration() of the Heart.
- AnyEntity() can stopPumping() for AnyReason(), and AnyEntity() can receive() support(). (What does AnyEntity() stop pumping? What kind of support does AnyEntity() receive and from whom?)
- The Heart can stopPumping() during a CardiacArrest.
- The Heart can receive() support from a HealthCareProvider.
- AnyResource() can activate() AnyMechanism(). (How does AnyResource() activate AnyMechanims()? Which mechanism does AnyResource() activate?)
- CPR can activate() BloodFlow.
- AnyLevel() can administer() AnyResource(). (What resource does AnyLevel() administer? How does AnyLevel() administer AnyResource()?)

- It requires an AdvancedLevel to administer() ElectricalActivity.
- AnyImpact() can determine() AnyLevel(). (How does AnyImpact() determine AnyLevel()? Why does AnyImpact determine level?)
- AdvancedLevel can determine() if HealthCareProviders KeepAlive the patient.
- AnyConstraint() can influence() AnyImpact(). (How does AnyConstraint() influence AnyLevel()? What level does AnyConstraint() influence?)
- PatientDeniesCPR influences whether or not the HealthCareProviders can KeepAlive the Patient.

Consider the use case for the life support SAP in Table 6.

- Use Case ID: 001
- Use Case Name: Support for life

Table 6
Use Case 1 for Life Support

Actor	Role
AnyParty	Patient, Health Provider

Class	Type	Attributes	Operations
Support	EBT	domain, name, context, supporterInfo, feedback, phoneNumber, satisfaction	provideAssistance()
AnyParty()	BO	serviceRequired, name, address, phoneNumber, email	requestVentilator(), diagnose(), stopCPR()

AnyMechanism()	BO	id, name, status, application, description, context, cost	restoreCirculation()
AnyReason()	BO	id, description, name, proof, limit, case	carrySymptom()
AnyType()	BO	id, property, subtype, interfaceList, methodList, typeName	regulateRespiration()
AnyEntity()	BO	entityName, type, position, state, status, isAlive, severityLevel	stopPumping(), receive()
AnyEvent()	BO	name, id, type, occasion, contract, contender	compete()
AnyResource()	BO	resourceId, resourceName, talent, category, means, natureOfResourc, requiredEfforts	activate()
AnyLevel()	BO	levelNo, name, type, amount, size, degree, volume	administer()
AnyImpact()	BO	id, name, level, consequence, force, power	determine()
AnyConstraint()	BO	restriction, type, severity, threshold, description	influence()

AnyParty(Patient)	IO	age, gender, dateOfBirth, medicalHistory, durationOfAilment, serviceRequired	request(), seekEncouragement()
AnyParty(HealthCareProvider)	IO	name, address, phoneNo, experience, reviews	provideTreatement()
AnyMechanism(CPR)	IO	id, name, status, application, description, context	improveHeartbeat()
AnyReason(CardiacArrest)	IO	cause, prevention, diagnosis, name, proof, limit, case	stopBloodCirculation()
AnyType(Physical)	IO	pressure, type, purpose, electricDose, portable, id	terminateImproperHeartbeat()
AnyEntity(Heart)	IO	heartRate, bloodFlow, cardiacCycle, status, isAlive, severityLevel	removeWastes(), beat()
AnyResource(BloodFlow)	IO	charge, field, potential, numberOfCells, waveType, resourceId, requiredEfforts	contractHeart()
AnyLevel(Advanced)	IO	levelNo, name, type, amount, size, degree, volume	scaleRespiration()
AnyImpact(Keep	IO	id, name, level, consequence,	effect(), sustain()

Alive)		force, power	
AnyConstraint(DNROrder)	IO	restriction, type, severity, threshold, description	cause()

Sequence diagram. This sequence diagram (Figure 8) shows a vertical dashed line or a solid line that represents the existence of an object over a period of time. The sequence diagram shows how the support life model flows over time.

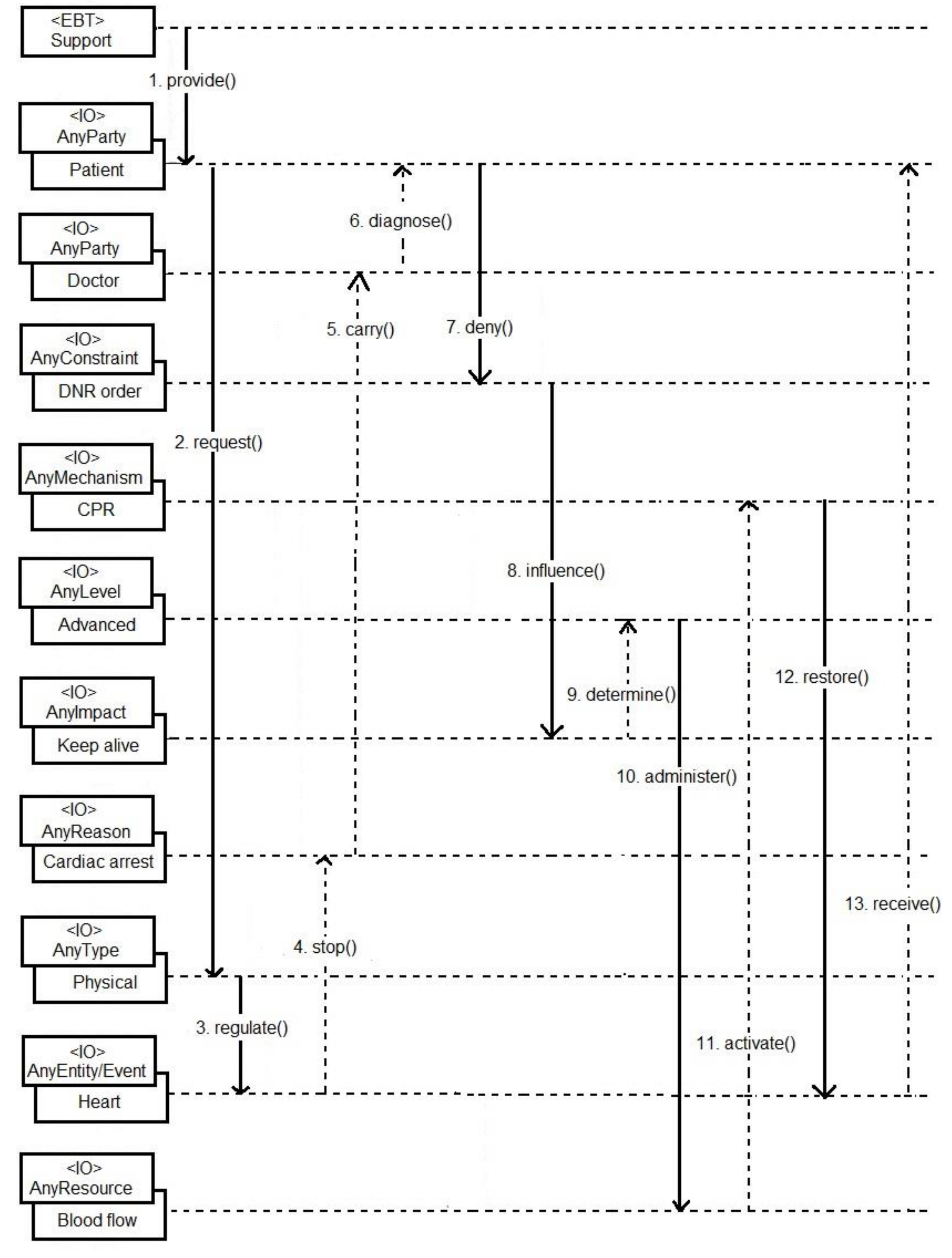


Figure 8. Sequence Diagram of Support in a Biological Setting.

Case Study 2: Providing Support for Children After Divorce

In a situation where parents (AnyParty) end their marriage (AnyReason) and where their child (AnyParty) needs support, the court may ask parents to provide periodic payments (AnyMechanism) for the maintenance (AnyEvent) of the child. This support helps the child to obtain proper care (AnyImpact). In many cases, each of the parents has an obligation to provide support for the child, even if the child does not live with either of the parents.

Child support comes in various forms of support such as financial, emotional, physical, and spiritual (AnyType). Sometimes, a parent shows negligence (AnyConstraint) and refuses to provide any kind of support to the child. During a divorce, certain factors need to be considered before determining the level of support each parent should give, such as how many children each parent has and the parent's total income. The money (AnyResource) parents provide is based on their income levels (AnyLevel) and existing financial status. The class diagram seen in Figure 9 graphically represents this problem.

Use case and description. Consider the use case for the support of a child after divorce in Table 7.

- Use Case ID: 002
- Use Case Name: Providing Support for Children After Divorce

It is important to consider certain questions with each class in the model.

- Support() maintains() AnyParty(). (What does Support() maintain? How is Support() maintained for AnyParty()?) For example, support maintains() the Child.

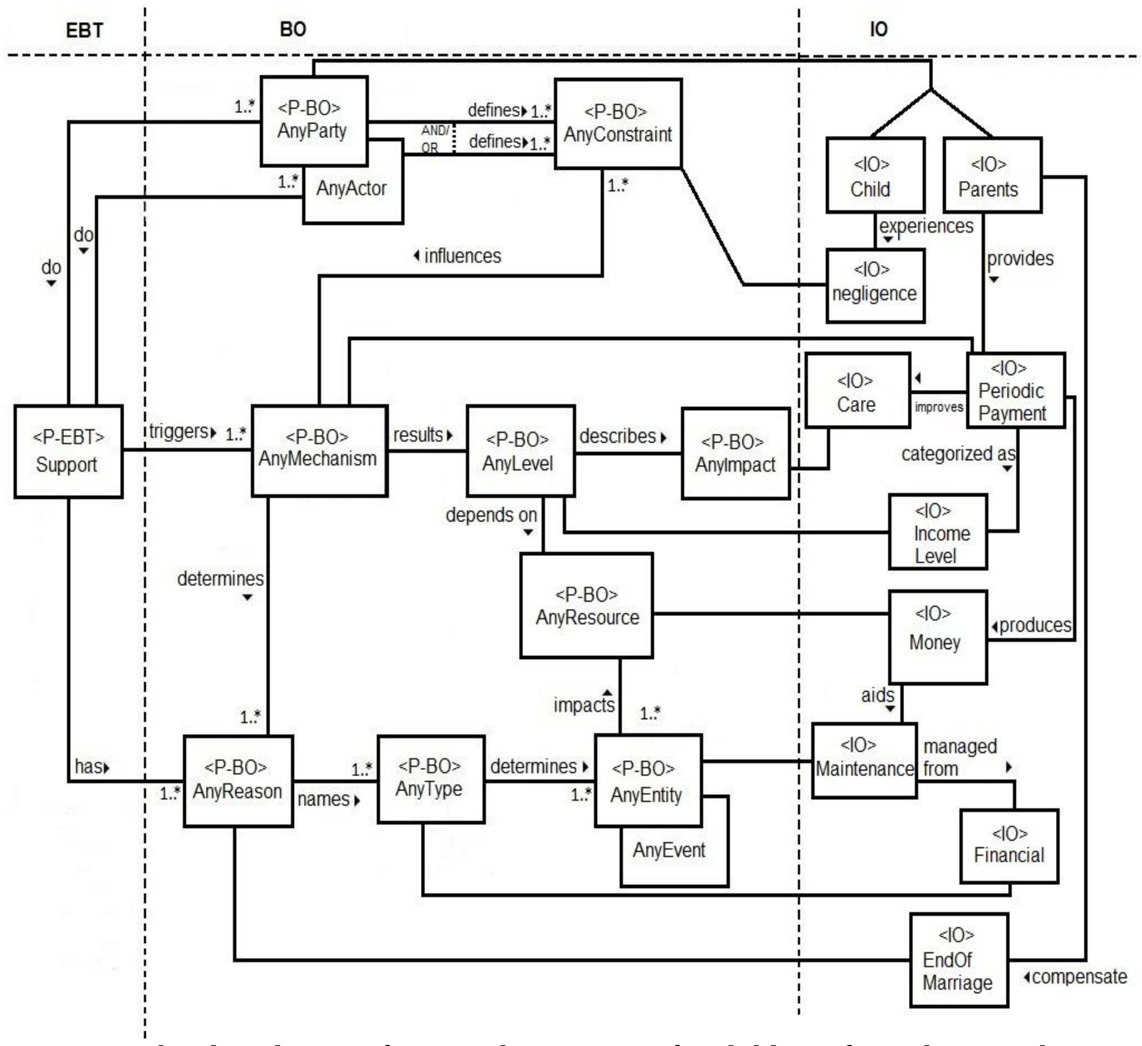


Figure 9. The class diagram for providing support. for children after a divorce, showing the EBT, BOs, and IOs.

- AnyParty() can receivePayment() from AnyResource(), and AnyParty() can getDivorce() for AnyReason(). AnyParty() in this case possesses() another AnyParty. (How does AnyResource() pay? What is the form of payment AnyParty() receives? How does AnyParty() getDivorced? Why does AnyParty() getDivorced? What is the other party that AnyParty() possesses?) For example, a child can receivePayment() in the form of Money, and the Parents getDivorce() because of EndOfMarriage.
- AnyResource() can handle() AnyEvent(). (What event does AnyResource() handle? How does AnyEvent() handle AnyResource()?) For example, Money can handle() Maintenance.
- AnyEvent() preserves() AnyParty(). (What does AnyEvent() preserve? How does it preserve AnyParty?) For example, Maintenance can preserve() the Child.
- AnyReason() leads() to AnyMechanism(). (What mechanism does AnyReason() lead to?) For example, EndOfMarriage leads() to PeriodicPayment.
- AnyMechanism() dependsOn() AnyLevel(). (What does AnyMechanism() depend on? How does AnyMechanism() depend on AnyLevel()?) For example, the PeriodicPayment dependsOn() IncomeLevel.
- AnyLevel() can determineClass() of AnyType(). (What class does AnyLevel() determine?) For example, the incomeLevel can determineClass() of Financial.
- AnyType() lacks() AnyConstraint(). (What does AnyType() lack? Why does AnyType() lack AnyConstraint?) For example, Financial lacks() Negligence.

- AnyConstraint() can influence() AnyImpact(). (How does AnyConstraint() influence AnyImpact()? What does AnyContraint() influence?) For example, Negligence can influence() Care.
- AnyImpact() affects() Support(). (How does AnyImpact() affect Support? What does AnyImpact() affect?) For example, Care affects() Support.

Table 7
Use Case-2 for Support

Actor	Role
AnyParty	Child, Parent

Class	**Type**	**Attributes**	**Operations**
Support	EBT	domain, name, context, supporterInfo, feedback, phoneNumber, satisfaction	maintain()
AnyParty()	BO	gender, name, address, phoneNumber, email	receivePayment(), getDivorce(), possesses()
AnyMechanism()	BO	id, name, status, application, description, context frequency	dependsOn()
AnyReason()	BO	description, name, proof, violence, case	leads()
AnyType()	BO	id, property, subtype, interfaceList, methodList, typeName	lacks()
AnyEntity()	BO	entityName, id, type, property, state, status, isAlive	separate()

AnyEvent()	BO	name, id, type, occasion, contract, contender	preserve()
AnyResource()	BO	id, resourceName, talent, category, means, natureOfResource, quantity	handle()
AnyLevel()	BO	id, name, type, amount, size, degree, volume	determineClass()
AnyImpact()	BO	name, level, consequence, force, power, implication	affects()
AnyConstraint()	BO	restriction, type, severity, threshold, description, duration	influence()
AnyParty(Child)	IO	age, gender, dateOfBirth, address, benefits, siblings	receiveBasicNecessities()
AnyParty(Parent~~s~~)	IO	names, dateOfMarriage, numberOfChildren, reasonForSeperation, occupation	seperatedSince(), compensate()
AnyMechanism(PeriodicPayment)	IO	amount, currency, status, duration, description, context, frequency	modeOfPayment()
AnyReason(EndOfMarriage)	IO	id, description, name, proof, violence, case	liveSeperately()
AnyType(Financial)	IO	id, Property, subtype, interfaceList, methodList, typeName	bearEducationFee()
AnyEvent(Maintenance)	IO	Quality, servicesProvided, benefits, name, occasion, contract, contender	control()

AnyResource(Money)	IO	amount, currency, bankAccount, category, means, natureOfResource, quantity	value(), save()
AnyLevel(IncomeLevel)	IO	id, name, type, amount, size, degree, numberOfCategory	distributeWealth()
AnyImpact(Care)	IO	id, name, level, consequence, force, power, implication	caterChildCare(), checkBackground()
AnyConstraint(Negligence)	IO	restriction, type, severity, threshold, proof, duration	causeHarm()

Sequence diagram. Figure 10 shows the sequence diagram for a child after his or her parents have gotten a divorce.

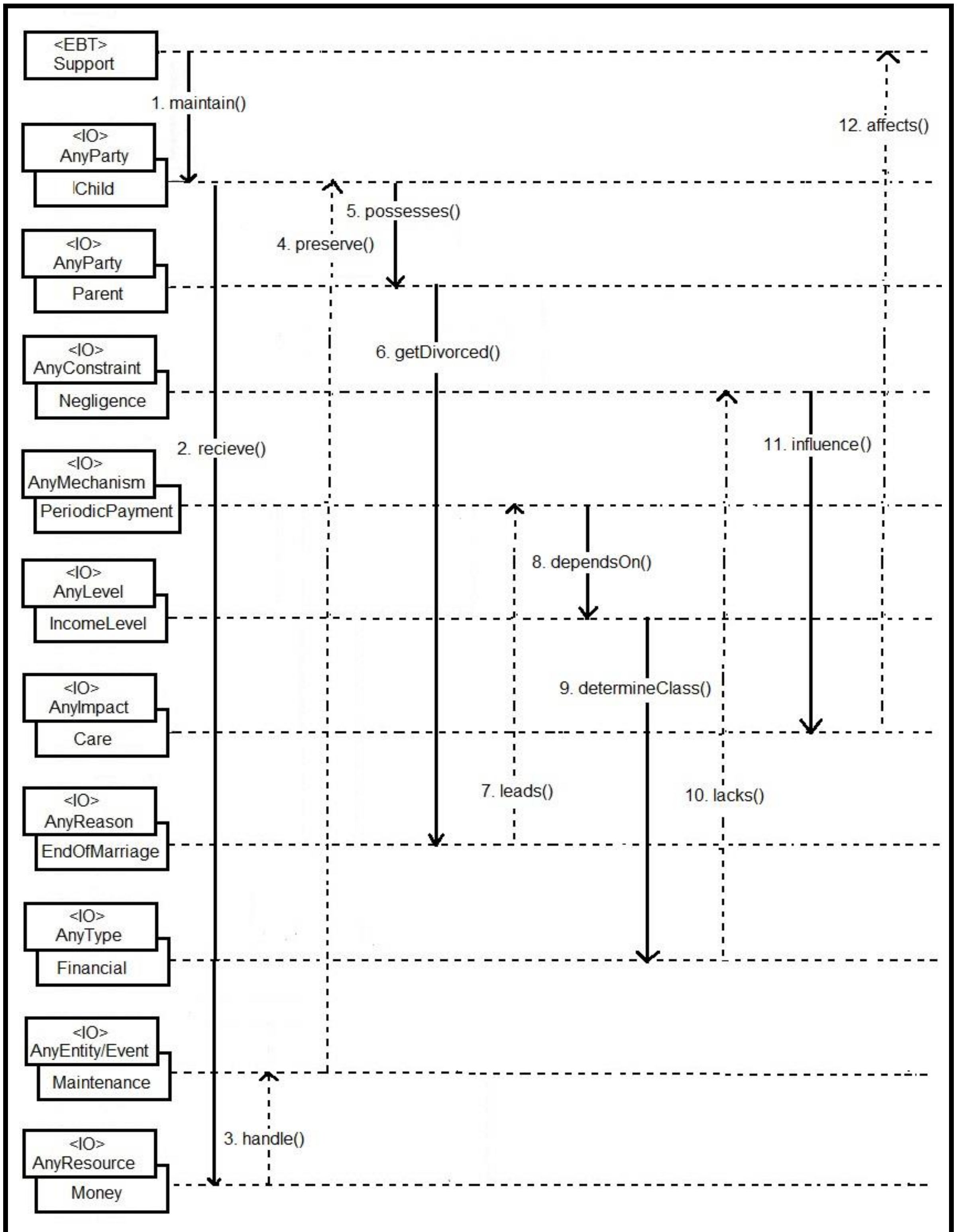


Figure 10. Sequence Diagram for Assisting Children After Divorce.

Comparative Study by Using Model Adequacies

Related pattern and measurability. Traditional patterns usually interact with the described stable pattern. It is important before creating a stable pattern to understand everything that needs to be included within the described pattern.

Related pattern. Figure 10 is a traditional pattern for a university software support system. Although it looks much simpler than the diagrams for the stable models, it has many disadvantages over the SSM.

Traditional models cannot be reused because they are specific to certain domains. They use only tangible objects or IOs. If human resource departments are to provide support instead of technicians, they will find the process more difficult if not impossible because the same traditional model cannot be reused. Stable models are reusable because they are created from EBTs and BOs. These are the core of the system and do not change even when the application domain changes.

Traditional models are created from tangible objects, so the entire application changes if trivial changes are made to the model. This is why the same traditional model becomes complex when we try to model different applications from the original model. With the stability model, all the diverse potential outcomes of the model are examined and considered prior to its creation. This means new models are more easily understood because the core model does not change.

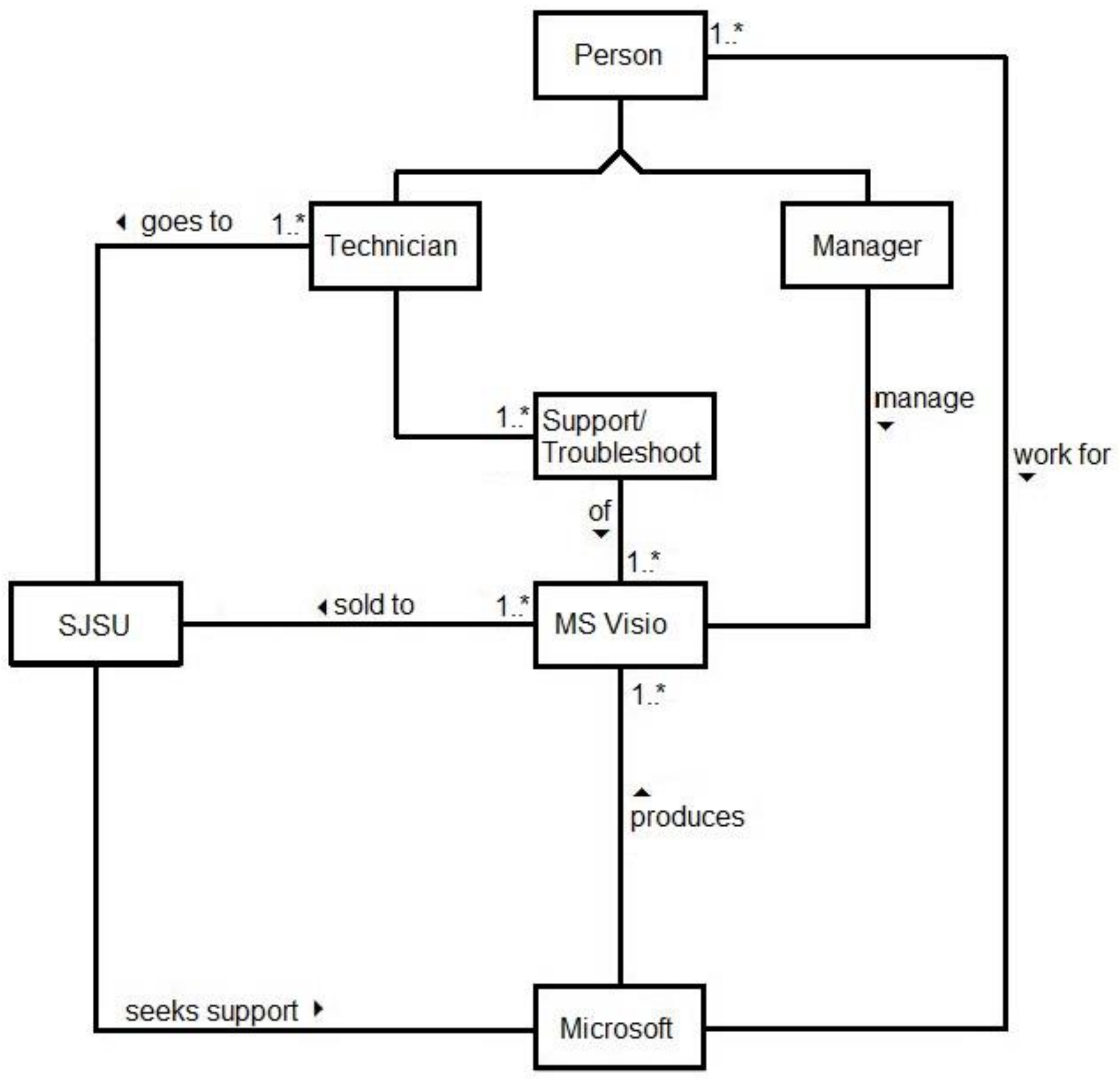


Figure 11. Traditional model of providing support for MS-Visio Software.

In a traditional model, maintenance is hard and costly. Although the initial setup is cheaper, each time a change is made developers must start over. As the project continues, the costs mount. In a stability model, maintenance is simple and inexpensive because the core knowledge of the domain doesnot change. Initially, the cost is more for development, but gradually the savings comes in the maintenance. The traditional model is not stable because the building blocks of a traditional model are the IOs, which are very unstable and tangible objects. The stability model is comprised of concepts such as EBTs, BOs, and IOs. EBTs are stable both internally and externally. BOs are also externally stable and internally adaptable. IOs are the outer links to the core system. Although these are less stable, they are not central to the model. Consider Table 8 for an overview of the benefits and disadvantages of each system.

Measurability. The quality of a project should always be of a high standard. It is important to sustain the flexibility factor in a model, allowing it to accommodate changes that might be made in the future. Reusability is another important factor because changes always occur. The two factors that make the stability model more effective are

- generalization, which is more effective because it focuses on concepts that last forever; and
- expressiveness, which allows the focus to be on the domain instead of the IOs.

The following formula can check the factor of reusability: $RF = C_u/T_c$. RF is the reusability factor; T_c is the total number of classes, and C_u is the number of classes reused. With the traditional model, RF = 0/7 = 0. This means the traditional model is less likely to be reusable because no class is reusable. With the SSM, RF = 12/12 = 1. This means everything can be reused.

Table 8
Support Comparison Between the Traditional Pattern and the SAP

Adequacy	Traditional Model	Stability Model	Weighted Score in %	
			Traditional Pattern	SAP
Reusibility	Specific to certain domains and deal with only tangible objects or IOs. Not reusable.	Use EBTs and BOs as an unchangable core. Reusable.	3	30
Ability to Understand	The traditional model needs to be changed as the application changes, which make them less understandable.	Everything is examined and the model does not need to be changed with the application, which makes them more understandable.	1	15
Maintenance	Difficult and costly.	Simple and inexpensive.	2	15
Cost	Initially cheaper but costs balloon as changes are needed.	Initially more expensive and more time-consuming, but the costs are low when changes are needed.	1	9
Stability	Unstable because it is built on IOs.	Stable because the core is built of stable EBTs and BOs.	0	30
	Total Score:		7%	99%

Note: SAP = stable analysis pattern.

As compared to the traditional model, the stability model contains fewer IOs. The stability model focuses mainly on EBTs and BOs, whereas the traditional model is completely dependent on IOs. The total operations formula calculates the qualitative measurability: $T_{OP} = C * O_{PC}$. Here, T_{OP} equals the total number of operations; C equals the total number of classes; and O_{PC} equals the operations per class.The number of classes and the operations define the simplicity of the project.

For the traditional model, we suppose five operations per class and therefore, $T_{OP}= 7 * 5 = 35$. With the SSM, we could suppose one, two, or three operations per class during the same scenario. $T_{OP} = 12 * O_{PC} = 15, 16, \text{or } 17$.

Increased numbers of operations with fewer numbers of classes will make a model simpler and more stable. Considering the above calculations, it is clear that the stability model has less complexity.

Summary

These detailed templates show how core knowledge of support can be used for numerous applications. This model can be extended to numerous contexts. Reusability, scalability, extensibility, interoperability, flexibility, unification, and adaptability are some of the biggest contributions of the SSM. A stable model of support built on the SSM is more reusable and more stable than the traditional model for support.

Compliance SAP

Pattern name. The compliance SAP is an EBT. The term compliance means yeilding to another person's request, charge, assignment, solicitation, or explanations. Other names for compliance are agreement, conformity, consent, and submission ("Compliance," 2016). The reason for choosing the term compliance for this pattern is that this term applies to almost all aspects of agreement, and it is more of a general term. The generality contributes to a more stable pattern.

The focal subject of compliance is participation, which is an EBT. In general, compliance can be between any numbers of individuals. However, it is important people comply on both the individual and team levels in order for work to be done effectively.

Context. When individuals or systems are in agreement with each other, one needs to comply with the rules declared by the other. Usually, compliance comes into all situations when one or more persons are working together to accomplish something . Consider the following scenarios as examples of compliance.

Comply with a celebrity. An advertiser (AnyParty) might use a celebrity (AnyParty) as a means of endorsement (AnyMechanism) for a product through television, magazines, or any other advertising medium. The Federal Trade Commission (FTC) provides useful guidelines (AnyRule) for endorsements and celebrity contracts (AnyAgreement) reflect these. The advertising testimonials and the celebrities giving them must follow the law (Compliance). There also be some restrictions (AnyType) involved in endorsing these products.

If it is a television advertisement, the advertisement would be shot with a videocamera (AnyMedia)and the footage would be telecasted. To keep an account of where a shot has its source on a tape, video crewmembers use video logging (AnyLog). Another method of advertising using a celebrity endorsement would be an endorsement letter (AnyVerification). Numerous fans (AnyParty) of the celebrity buy (AnyOutcome) the product advertised because their favorite celebrity (AnyReason) is endorsing the product.

Complying in a prison. A prisoner (AnyParty) complies with the guards' (AnyParty) orders. Although prisons are not supposed to use torture, certain types of psychological torture are used to ensure prisoner compliance, such as solitary confinement. In general, psychological torture does not physically harm the victims, but they may lose their desire to live. All the prisoners and guards conform (AnyOutcome) to their respective social roles especially the guards. The guards have a contractual agreement (AnyAgreement) with the prison for compensation that includes the legal aspects and rules they must follow (AnyRule). There is a prisoner verification system (AnyVerification), which prevents prisoners from collecting government aid while in prison. The guards can access the policies (AnyMedia) of the prison because the information is stored in the prison records (AnyLog). Prisons can have minimum or maximum (AnyType) security.

Problem

As of now, there is no SAP existing for compliance that defines the core knowledge of compliance. Once there is an SAP for compliance, others can build as many applications as they want, and it can be used repeatedly without changing its core. However, there are certain requirements necessary to build the SAP.

Functional requirements.

- Compliance can have any number of actors involved inthe agreement. These actors comply with framed rules. Compliance is the underlying theme to develop the SAP.
- AnyParty uses of the system as a legal entity. AnyParty is not involved in framing the protocols or other leadership roles. AnyParty generally has a name, contact information, and at least two roles. An employee of an organization participates() in the company projects and can playRoles() on his team by collectingData() and can interact() with other employees.
- AnyActor is in the system or actors, who differ from the legal users. AnyActor have a name, a unique ID, a role, and a category. It has operations such as playRole(), interact(), request(), explore(), and receive().
- AnyRule depends on the scenario. AnyActor or AnyParty creates AnyRule based on the environment. AnyRule has a name, a unique ID, and a description. Others must follow() AnyRule, and the government must allow() it. For example, a rule must be accepted() by higher authorities to be followed() and controlled().
- The user follows AnyMechanism to create agreement that results in AnyOutcome. It has attributes such as context, id, and mechanismName. AnyMechanism uses operations such as execute(), activate(), and attach().

- AnyOutcome is the end result of compliance. AnyOutcome has an id, a system, and an approach. It has operations such as access(), connects(), and directs().
- AnyVerification confirms AnyOutcome. Some attributes of AnyVerification are id, context, and methodology. AnyVerification operates using verify(), check(), and status().
- AnyReason is influenced by conditions or requirements. AnyReason can cause AnyActor to comply with the agreement. AnyReason can result in AnyTypeAnyReason often has a description() to explain the details of the reason. Once the details are understood, one can decide() what the conclusion should be.
- AnyAgreement can be in any form—verbal or oral. It has attributes such as id, members, and category. It has operations such as status(), playRole(), and interact().
- The result of AnyReason can be AnyType. AnyEntity has AnyTtpe. AnyType has a name, an id, a property and possibly a subtype. It has operations such as change(), operateOn(), subtype(), and classify(). For example, cars are classified() into several subTypes() based on what they operateOn(), and these types change() over a period of time.
- AnyMedia defines compliance, and AnyMedia is archived in AnyLog. AnyMedia has a name, id, and category. AnyMedia connects() and gives access() to compliance. AnyMedia helps to broadcast() various compliances and to display() them to others.

- AnyLog records AnyAgreement for future refernce using AnyMedia. AnyLog has attributes such as id, name, numberOfEntries, and references. AnyLog in a store could be an excel sheet where transaction details are stored(). AnyLog allows users to search() for any details of previous compliance. If it is a spreadsheet, the user can insert() or extend() items into that log.

Non-functional requirements.

- The created protocol should be relevant in that it fits into the context of the scenario. If, for example, the protocol requires electronic storage when only paper storage is available, then it is not useful. Data should match the scenario. For example, employees must comply with regulatory commission rulings. Relevant laws and regulations imposed by the organization would follow these. If the industry is making computers and the regulations posted are for fishing, they are not relevant.
- The users must understand the agreement before they can comply. For example, COPPA lists the regulations for popular online services aimed at children (Liccardi, Bulger, Abelson, & Weitzner, 2014). However, children, in general do not understand these regulations. In addition, many websites are created by novice users who are unfamiliar with the regulations. It is difficult to comply with COPPA when it is not accessible to the average user, and it is difficult to regulate compliance when those who need to report discrepancies do not understand the requirements..

- Framed rules or agreements should be justifiable to help defend the client in case of any arguments. Without justification for rules, users are less likely to follow them. One example of justifiable laws are those laws that stipulate drivers follow traffic signals. It is easy to recognize the hazards at road intersections and junctions when traffic signals are not obeyed. Justification facilitates compliance.
- Compliance should be scalable. It must change by itself or update its credentials. There should be provisions to scale all the predefined protocols. In general, the laws change over the time. Scalable compliance changes with them. Consider a physician and patient. The patient falls asleep immediately after taking his medicine and so the physician changes the prescription. Compliance policies should be scalable according to the current needs and demands.

Challenges and Constraints

Several scenarios illustrate some of the challenges associated with compliance. For example, when the initial rules are set up, theype of client (AnyActor) and server (AnyActor) should be identified properly to frame the set of rules (AnyRules). It is very important to match user with system when it comes to computer networks (AnyMedia) because of their compatibility issues. If atarget system or model is not understood then it leads to improper implementation and potential failure. One should conduct a proper and in-depth study to discern the type of environment where the system operates.

Or consider a device (AnyMedia) connected to the Internet (AnyMedia). Several online threats exist that cause the system to fail. Without frequent updates to keep ahead of changing threats, the system fails from security concerns. A provision to update the framed protocols (AnyRules) and secure the system (AnyActor) must be in place. Newly created or designed rules should be updated in the server periodically in a way the client can access them. Newly framed rules should not create system problems.

It is very hard to frame the rules (AnyRules) without having any idea of a network (AnyMedia) and the rules that are to be implemented for a secure computing. In another scenario, suppose a computer designer must design an online system that has high-performance and high security. The designer considers all the factors such as number of users, complexity of the system, and network security protocols before designing the system. However, the designer constantly starts over with every error the system has. In order to analyze the behavior of a system, the designer should consult and learn from all the log errors. Monitoring errors enables the designer to learn from them and comply with customer demands.

Interacting with client (AnyActor) also requires compliance. For instance, in order to interact with and convince a client regarding a health insurance policy, the agent should follow proper compliance. Following set protocols for policy explanation, claim filing, and customer service allows the agent to build trust with clients and help them recognize the value of the company's insurance policy (AnyMedia). To assure positive interaction, all members of the company interacting with clients should comply with company regulations regarding these interactions.

In addition, the client (AnyActor) must understand policwhen they enroll for the insurance plan. Consider a smoker who takes out a policy for a non-smoker. When he tries to make a claim on the policy, the claim is denied because he did not comply with the rules. A client comply with the rules (AnyRule) only when he or she has understood them clearly. The agent should present the rules in such a waythat the agent is sure the client understands them. One of these ways is using videos or images to aid comprehension. Another option would be asking the client questions as well as allowing the client to ask questions about the policy.

There are certain rules (AnyRules) that are framed by search engines such as Google (AnyActor) for listing websites. Businesses (AnyActor) should follow the rules to ensure proper exposure from search engines. A team should gather information from several sources to ensure their website is optimized for display in the search engine when keywords are entered.

Another important factor of online business interactions is the signed privacy policy (AnyRule). This policy is frequently long and repetitive and the user (AnyActor) of the system may or may not understand it. In addition, policies frequently change and often users are not informed of the changes. This could create future problems with the user. For instance, if the user has an account on a social networking website and the website updates the privacy terms stating that they may use the user's personal information for analysis and marketing, but the change is not sent to the user, it may cause trouble when the user discovers it. The company's (AnyActor) reputation may be at risk and legal issues could go to court. Companies should provide notifications to users when updating policies.

Another example of this is when user signs up for a social media website, reads the terms, (AnyMedia) and accepts them. Later the user may dislike the system

(AnyActor) and take the company to court over it. The company may state that the user can not file a case in court due to the conditions initially signed. However, after time, the user may have forgotten these conditions or the user may have changed in such a way as to no longer agree to the conditions. To avoid such unforeseen problems, the signed agreement should be stored (AnyLog) in order to solve all future problems. Then, the user have access to the terms and can review them before he or she files a complaint. See Table 9 for a summary of these scenarios.

Table 9
The Challenges and Constraints of Sample Compliance Scenarios

Title	Problem	Constraint	Solution
Client Server Computing.	It is important to match clents and servers effectively	Identifying a target model.	Conduct an in-depth study on the situation before placing the system.
Minimizing Online Threats.	Connecting to the Internet presents constantly changing threats to devices.	Protocol document modification.	Protective software needs to be updated authomatically and frequently.
Designing Online Systems,	All the factors need to be considered when designing a new system for security and performance.	Protocol and design issues.	Keeping accurate logs and reviewing the errors contained in them improves design.
Making Websites Available to Search Engines,	Online content must follow certain guidelines in order to be listed in search engines.	Following seach engine optimization rules and regulations.	Gathering guidelines from several sources and designing a website based on all the information result in better visibility.
Social Networking Privacy Policy.	Rules are frequently updated without user notification.	Need to understand rules.	Guidelines should be posted in a way that all users are aware of

			changes.
Client Misunderstands Policy Information	Client purchases a health insurance policy but is uncertain what it covers.	Clarify policy information.	The insurance agent should go over the policy with the client prior to purchase and encourage the client to ask questions.
Client Interaction Strategies	Insurance salespeople are generally mistrusted.	Building trust with insurance agents.	Creating solid guidelines improves the client-salesperson relationship and build trust.
Storage of User Contracts	Contracts signed online are frequently stored digitally and subject to system failure issues.	Documents should have more than one storage facility.	Documents can be printed and stored separate from the computer.

Constraints.

- Compliance should use AnyMechanism to create one or more agreements.
- Compliance should have one or more reason to have an agreement.
- AnyParty is a legal user and can define AnyRules.
- AnyParty agrees to comply.
- AnyActoris the user apart from legal users and agrees to comply.
- AnyActor provides details for one or more agreement.
- Compliance uses AnyMechanism.
- AnyMechanism uses AnyAgreement.
- AnyMechanism results in AnyOutcome.
- AnyOutcome can undergo AnyVerification.

- AnyParty creates AnyAgreement, and AnyMechanism follows AnyAgreement to achieve AnyOutcome.
- AnyVerification follows AnyRule to verify AnyOutcome.
- AnyReason is the reason for compliance and has AnyType. It is used to create AnyAgreement.
- AnyType has AnyOutcome and is recorded in AnyMedia.
- AnyMedia records AnyLog.

Solution

We provide a workable solution by forwarding a robust and extendable compliance SAP that uses SSM concepts. The given solution provides a compliance pattern, which allows one to build numerous applications on cross-platform domains.

Pattern structure. The relation between the EBTs and BOs within the Compliance SAP is shown in Figure 12.

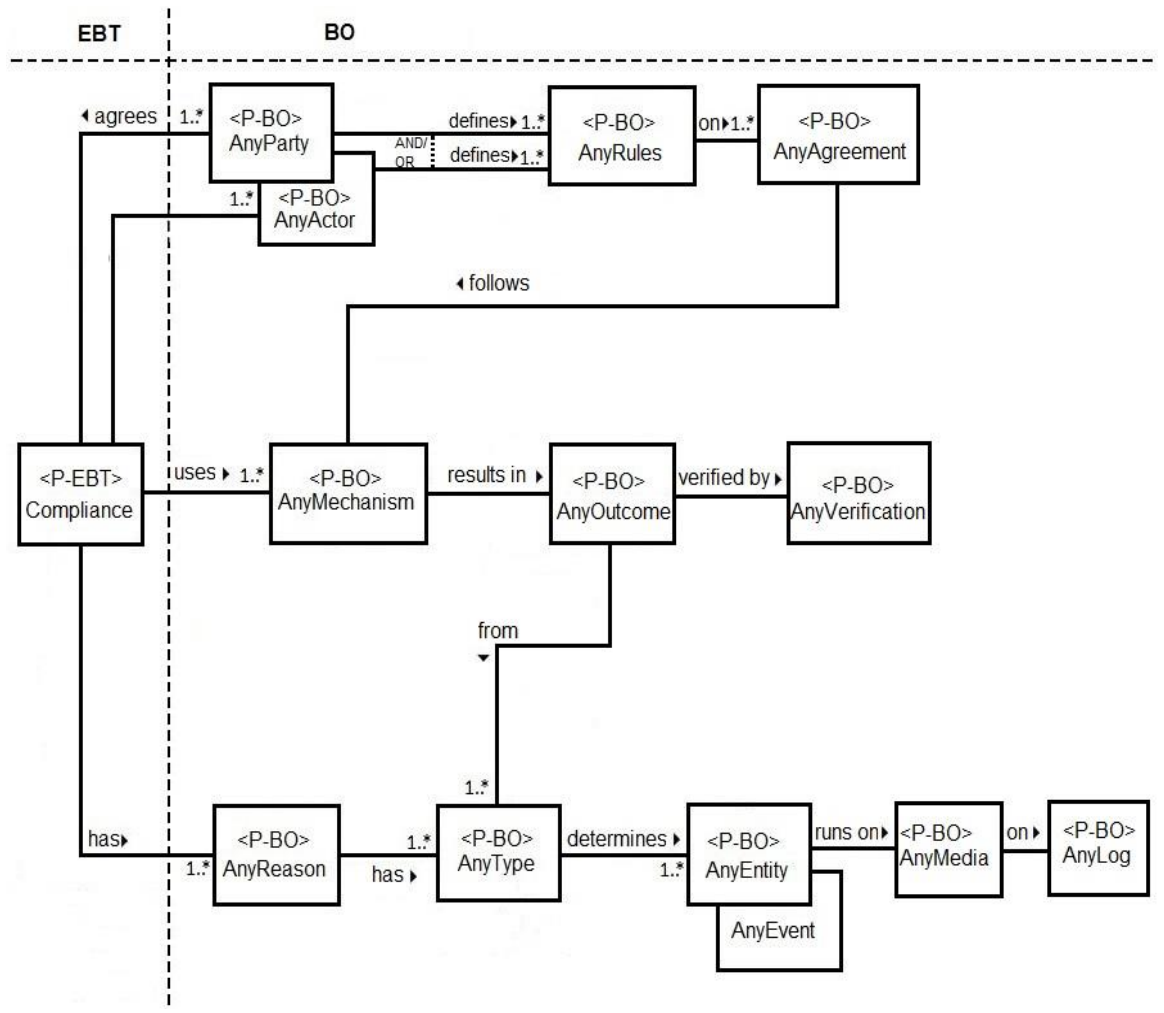


Figure 12. Class diagram of the compliance SAP showing the EBT and BOs.

Class diagram description.

- AnyParty or AnyActor agrees to compliance and defines AnyRules.
- Compliance uses AnyMechanism and results in AnyOutcome.
- AnyVerification is from AnyType and verifies AnyOutcome.
- AnyReason has AnyType.
- AnyMedia is of AnyType.
- AnyLog resides on AnyMedia.

Pattern structure. The compliance SAP consist of the following participants:

Classes.

Compliance represents itself. This class is expected to consist of the attributes and operations that manage real compliance process. This class has attributes such as type, context, and hasConsent. The compliance class has operations such as agree(), adhereRules(), and obeyOrders().

Patterns.

- AnyParty represents those handling compliance. It accommodates all parties that are a part of the compliance process. AnyParty can be a person, organization, country, or political party. AnyParty has attributes such as name, id, and location. It has operations such as command(), authorize(), and request().
- AnyActor represents the actors that are involved in compliance. An actor could be software, hardware, a person, or a creature. It has attributes such as name, id, and type. AnyActor has operations such as instruct(), giveDirection(), and organize().
- AnyMechanism represents the mechanism used to perform compliance operations. It has attributes such as name, context, description, application, and status. It has operations such as execute(), attach(), detach(), and activate().
- AnyType represents the different types of compliance needed by an actor or a party. Some types of compliance are medical compliance, physical compliance, and financial compliance. AnyType has attributes such as id, name, interfaceList,

methodList, and property. The operations are change(), categorize(), and subtype().

- AnyReason represents the reason compliance is required. It could be a fact, a situation, an explanation, a rationale, or a condition. It has attributes such as description, proof, and justification. It has operations such as conclude(), examine(), and resolve().
- AnyAgreement represents being in accordance with the compliance. This class has attributes such as documentName, isApproved, and transactionType. It has operations such as negotiate(), agree(), and settle().
- AnyRules represent the set of guidelinesneeded to accomplish compliance. It impacts the assessment criteria of compliance. This class has attributes such as ruleName, numberOfRules, and isLawful. It has operations such as control(), takeover(), and overrule().
- AnyOutcome is the result of AnyParty or AnyActor's compliance. This class has operations such as conclusion, result, and issue. It has operations such as achieve(), complete(), and terminate().
- AnyVerification represents the method of verifying AnyOutcome of the compliance.   This class has attributes such as verificationNumber, methodToVerify, and typeOfVerification. This class has operations such as authenticate(), check(), confirm(), and validate().
- AnyMedia represents the media through which compliance takes place. Certain media allow searches that improve access to and understanding of

AnyAgreement. It has operations attributes such as name, availability, and isAvailable. It has operations such as display(), store(), capture(), broadcast(), and connect().

- Compliance occurs for AnyEntity. AnyEntity is named by AnyType of compliance. It has attributes such as id, name, and status. Its operations can be to update(), relationship(), and type().
- AnyEvent: Compliance is done for AnyEvent and is named as AnyType of compliance. It has attributes such as name, occasion, type, and outcome. Its operations can be to appear(), to happen(), and to arrange().
- AnyLog represents record keeping. It could be a file, digital file, or anything that can be stored. AnyLog allows us to search for a specific compliance and has attributes such as id, name, numberOfEntries, and reference. It has operations including search(), insert(), and extend().

CRC Cards

Table 10
CRC Card Information for the Compliance SAP

Participant	Duty	Attributes	Collaboration	
			Client	Server
Compliance (EBT)	To agree	type, context, id, name	AnyParty, AnyActor, AnyMechanism, AnyReason	agree(), adhereRules(), obeyOrders()
AnyActor (BO)	To be invovled in compliance	age, name, id, gender, email, phoneNumber	Compliance, AnyRules	instruct(), giveDirection(), organize()

AnyParty (BO)	Represent compliance handler	location, id, phoneNumber, email	Compliance, AnyRules	command(), authorize(), request()
Any Mechanism (BO)	Generate results	name, context, application, description, status	AnyAgreement, Compliance, AnyOutcome	execute(), attach(), detach(). activate()
Any Agreement (BO)	To represent harmony	id, name, detailsm, type, isLegal	AnyRules, AnyMechanism	chooseAgreement(), selectMechanism(), userDefinesAgreemen t()
AnyRules (BO)	To set guidelines	rule, number, createdBy, eligibleFor, isLawful	AnyParty, ANyAgreement	control(), takeover(), overrule()
Any Verification (BO)	To verify the outcome	number, method, details, isCorrect, type	AnyOutcome	authenticate(), check(), confirm(), validate()
Any Outcome (BO)	Mechanism outcome	id, conclusion, details, index	AnyMechanism, AnyVerification	achieved(), complete(), terminate(),
AnyReason (BO)	Specify details	id, description, justification, proof, context	Compliance, AnyType	analyze(), explain(), defend()
AnyMedia (BO)	To display	id, name, type, capability, entry, securityLevel	AnyEntity, AnyEvent, AnyLog	display(), store(), capture(), broadcase()
AnyLog (BO)	To store records	id, name, number, size, location, references	AnyMedia	search(), insert(), extend(),
AnyType (BO)	Designate category	id, property, interfaceList, methodList, subtype,	AnyReason, AnyEntity, AnyEvent,	classify(), operateOn(), attach(), detach()

		typeName	AnyOutcome	
AnyEntity (BO)	Exist independ-antly	entityName, id, type, position, state, status, isAlive	AnyType, AnyMedia	performFunction(), relationship(), new()
AnyEvent (BO)	To happen or occur	name, id, type, occasion, contract, contender	AnyType, AnyMedia	compete(), celebrationFor(), rank()

Consequences

The pattern provides notable outcomes that help it in achieving its enduring goals. They are given below.

- The pattern reduces complexity. The compliance pattern modularizes different viewpoints that are included in the compliance procedure and this makes the pattern very easy to understand.
- The pattern allows the inclusion of new IOs. It does this by taking into account the application area. It gives the pattern's clients a stable center model that incorporates all the vital highlights of compliance.
- The pattern provides flexibility and scalability. It can be applied to different compliance domains and modified.
- The pattern mproves adaptability and extensibility. It is versatile. It handles the expansion for new, sophisticated highlights.

Applicability

Table 11
Applicability of the Compliance EBT Across Several Disciplines

BO	Buying	Ad	Prison	Finacial	Medical
AnyParty	Salesperson	Celebrity, buyer	Guard, prisoner	Owner, client	Doctor, patient
Any Mechanism	Persuade	Endorse	Torture	Invest	Medication
AnyReason	Increse sales	Promote	Confess	Money	Care
AnyType	Free Trial	Restrictive	Psych- ological	Unit	Prescription
AnyEntity	Item	Product			Medicine
AnyEvent			Crime	Business	
Any Agreement	Form	Contract	Contract	Prospectus	Decision
Any Verification	Third-Party	Letter	Prison System	Prospectus	License
AnyRule	Business Laws	FTC Guidelines	Prison Rules	Investing Laws	Medical Laws
AnyMedia	In-person	Television	Policies	Stock Exchange	Prescription
Any Outcome	Item sold	Product purchased	Prisoner Confesses	Profit	Treatment
AnyLog	Daily sales records	Video Log	Prison Records	Policy Files	Medical Records

Case Study 1: Compliance by Salesperson

Scenario. When a buyer visits a store, he or she might come across a salesperson. Assume the salesperson offers a free product trial. For example, if the item is a bag of frozen french fries, the salesperson may offer a few cooked fries to the buyer to sample. This increases the chances of selling that product. After trying the product, the seller attempts to persuade the buyer to purchase the item.

This is an effective selling technique because there is a direct face-to-face interaction with the buyer, the buyer is allowed to try the product without risk, and their feedback can be recorded even if they do not like the product.

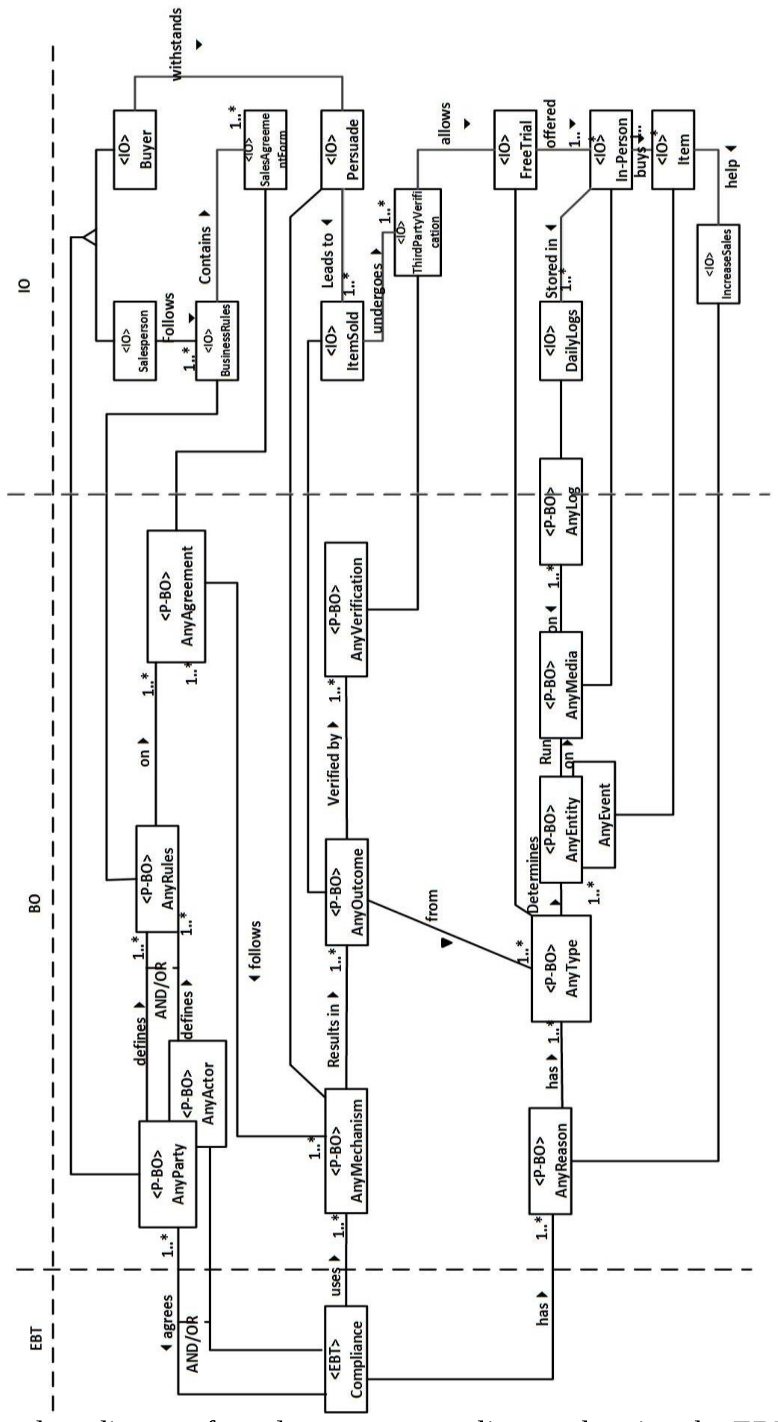


Figure 13. The class diagram for salesperson compliance. showing the EBT, BOs, and IOs.

Use Case-1 for Compliance

Use case and description. Consider the use case for the compliance between a salesperson and buyer in Table 12.

- Use Case ID: 001
- Use Case Name: Providing Compliance Between a Salesperson and Buyer

Table 12
Use Case-1 for Compliance

Actor	Role
AnyParty	Salesperson, buyer

Classes	Type	Attributes	Operations
Compliance	EBT	name, type, details, validThroughDate, owner	agree()
AnyParty()	BO	id, name, type, role, member	agreesToComplyWith()
AnyActor()	BO	id, name, type, role, department	participate()
AnyRules()	BO	name, id, detailsdescription, validThrough	identify()
AnyAgreement()	BO	name, type, description, expiryDate, signingAuthority	uses()
AnyOutcome()	BO	id, nme, type, description, implication	analyze()
AnyMechanism()	BO	name, id, description, mode, method	execute()
AnyVerification()	BO	type, name, time,	startProcess()

		description, verificationEfforts	
AnyReason()		id, type, description, domain, name	identify()
AnyType()	BO	id, name, properties, subtype, department	worksOn()
AnyMedia()	BO	id, name, mode, type,method	connect()
AnyLog()	BO	id, name, type, description, list, date	stores()
AnyEntity()	BO	name, id, type, description, price	performFunction()
AnyEvent()	BO	name, id, type, occasion, contract, contender	compete()
AnyParty (Buyer)	IO	id, name, type, role, member	Withstands()
AnyParty (Salesperson)	IO	id, name, type, role, member	Follows()
AnyRules (BuinessRules)	IO	name, id, list, description, validThrough	contains()
AnyAgreement (SalesAgreementForm)	IO	name, type, description, expiryDate, signingAuthority	containedBy()
AnyOutcome (ItemSold)	IO	id, name, type, description, implication	undergoes()
AnyMechanism (Persuade)	IO	name, id, description, mode, method	leadsTo()
AnyVerification (ThirdPartyVerification)	IO	type, name, description, time, verificationEfforts	allows()
AnyReason (IncreaseSales)	IO	id, type, name, description, applicableDomain	helpedBy()
AnyType (FreeTrial)	IO	name, properties, subtype, department, id	offers()

AnyMedia (InPerson)	IO	id, type, name, mode, method	offeredBy()
AnyLog (DailyLogs)	IO	id, name, type, description, date, logList	storedIn()
AnyEntity(Item)	IO	name, id, type, description, price	help()

- AnyParty() agrees to comply using AnyMechanism(). (Who agrees to comply? What mechanism is used to comply?) A buyer agrees to buy a product. The salesperson persuades a shopper to buy a product.
- AnyParty() agrees to comply with AnyRule(). (How strictly are the rules followed? What happens when these rules are broken?) The salesperson complies with business rules.
- AnyActor() participates in the compliance process (Who participates in the compliance process, and why?) The salesperson participates in the compliance process.
- AnyRule() identifies AnyAgreement() (What identifies AnyAgreement?) (How is AnyAgreement() identified?) Business rules guide a sales agreement form.
- AnyAgreement() uses AnyMechanism(). (What uses any AnyAgreement()?) Persuasion leads to a buyer signing a sales agreement form.
- AnyOutcome analyzes AnyMechanism(). (What analyzes AnyMechanism?) The distribution of free samples is analyzed by the number of sales after free samples are given.
- AnyMechanism() executes the methodology by following AnyAgreement. (Who follows methodology?) Persuasion leads to item being sold.

- AnyVerification verifies compliance occurred. (Who verifies the process of compliance?) Third-party verification examines free trial results.
- AnyReason identifies AnyEntity or AnyEvent(). (What does AnyReason identify?) An increase in sales can be achieved by finding more buyers.
- AnyType() works on AnyLog(). (What does AnyType work on?) All free trials and the sales resulting from them are recorded for later review.
- AnyMedia() connects AnyEntity() or AnyEvent() to AnyActor or AnyParty() through AnyMechanism(). (What does AnyMedia() connect through AnyMechanism()?) Television connects people and products through advertisements.
- AnyLog() stores information about AnyEntity or AnyEvent(). (Where is the information about AnyEntity AnyEvent() stored?) Information about the company products is stored in digital records.
- AnyEntity() performs a function. (What function does AnyEntity() perform?) The french fries taste good and make the customer happy.
- AnyEvent() allows AnyParty() to compete(). (Whom does AnyEvent allow to compete?) The salesperson does a promotion in the grocery store so his product can compete with other offered products.

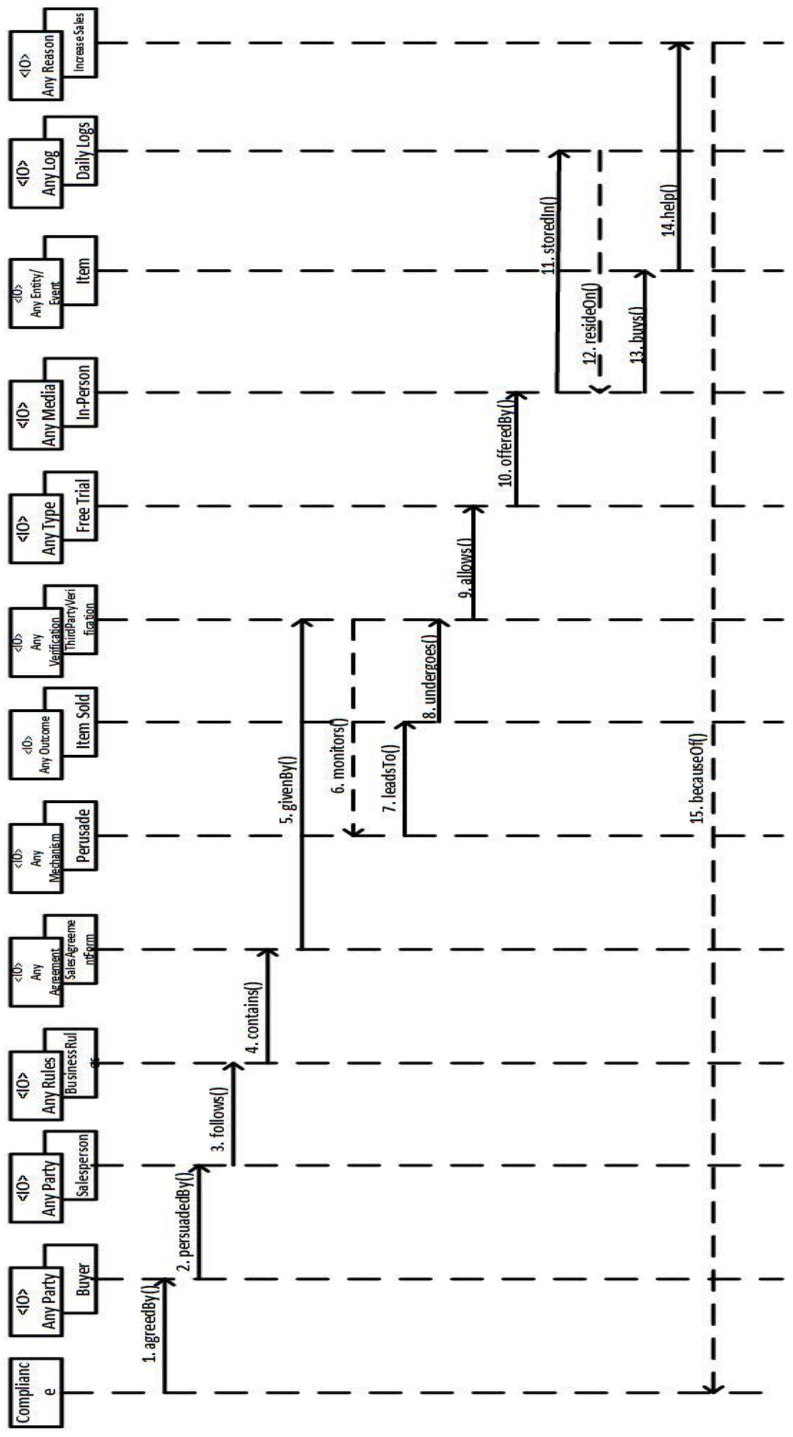


Figure 14. Comply with Buying

Case Study 2: Complying With a Physcian’s Prescription

This case study is about medicinal compliance when a physician (AnyParty) prescribes medicine (AnyEvent) to a patient (AnyParty) as a part of the treatment process. In all hospitals and clinics, the physician, after examining the patient, may prescribe medicines (AnyEvent). The patient adheres to the physician’s directions (AnyAgreement) and follows the prescription (AnyType) information. In this case, only the physician makes the decision (AnyAgreement) and the patient obeys. In some cases, the physician and the patient may make the decistion for a treatment (AnyOutcome) together.

Not following doctor recommendations is a serious problem (non-compliance). In some cases, the patient consumes the medicine for self-care (AnyReason). The World Health Organization reports indicate that over 50% of the patients from developed countries fail to adhere to suggestions given by their doctors. The physician needs to be licensed (AnyVerification) to make decisions and is governed by laws for medical practice (AnyRule). The physician and patient interact face to face and the prescription is written on paper or transmitted electronically (AnyMedia). Information about the prescription is stored in hospital records (AnyLog).

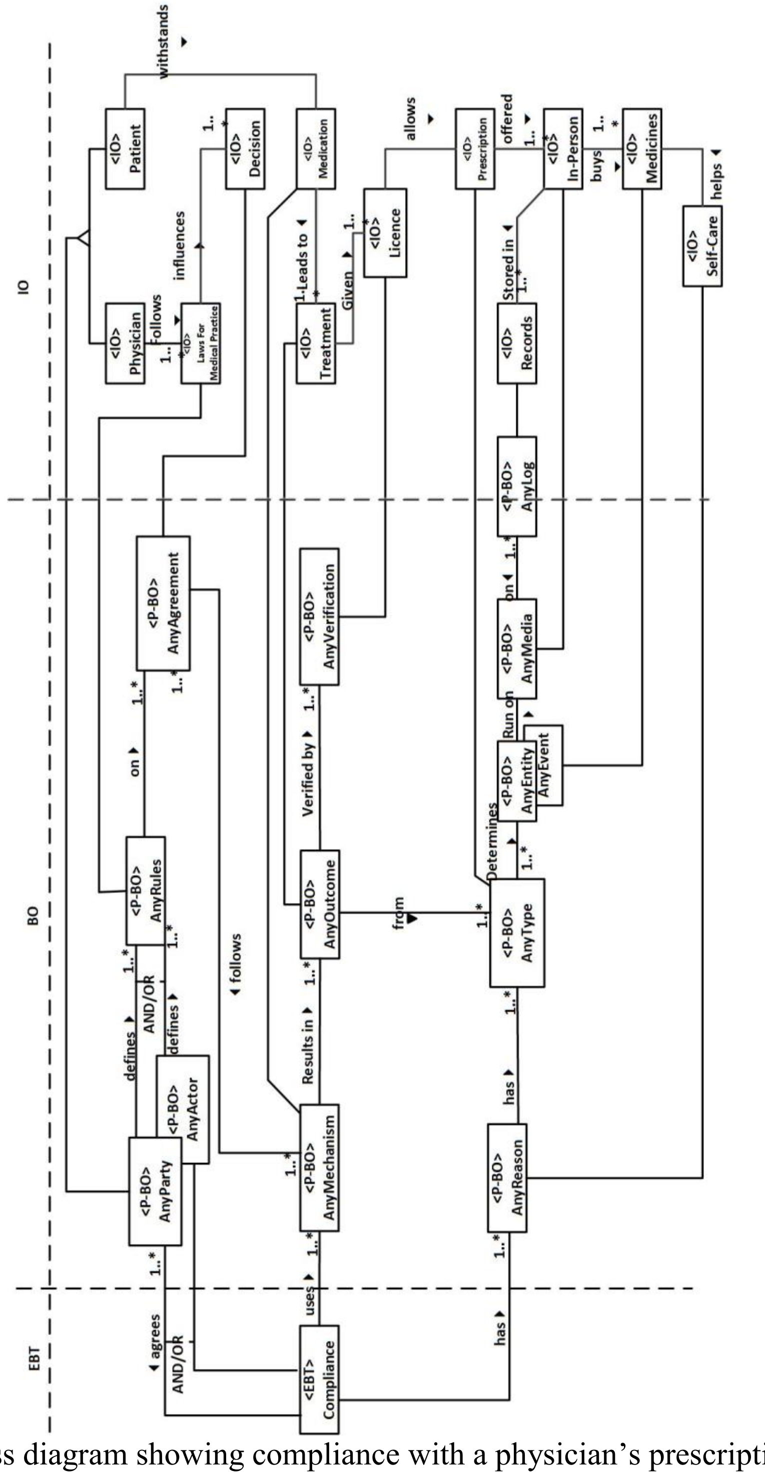


Figure 15. Class diagram showing compliance with a physician’s prescription, showing the EBT, BOs, and IOs..

Use case and description. Consider the use case for the compliance with a physcian's prescription in Table 13.

- Use Case ID: 002
- Use Case Name: Complying With a Physician's Prescription

Table 13
Use Case-2 for Compliance

Actor		Role	
AnyParty		Doctor, patient	
Classes	Type	Attributes	Operations
Compliance	EBT	name, type, details, validThroughDate, owner	comply()
AnyParty()	BO	id, name, type, role, member	frameRules()
AnyActor()	BO	id, name, type, role, department	initiateAgreement()
AnyRules()	BO	name, id, detailsdescription, validThrough	update()
AnyAgreement()	BO	name, type, description, expiryDate, signingAuthority	follow()
AnyOutcome()	BO	id, nme, type, description, implication	conclude()
AnyMechanism()	BO	name, id, description, mode, method	iterate()
AnyVerification()	BO	type, name, time,	identifyFlaws()

		description, verificationEfforts	
AnyReason()		id, type, description, domain, name	displayType()
AnyType()	BO	id, name, properties, subtype, department	catagorize()
AnyMedia()	BO	id, name, mode, type,method	broadcast()
AnyLog()	BO	id, name, type, description, list, date	search()
AnyEntity()	BO	name, id, type, description, price	status()
AnyEvent()	BO	name, id, type, occasion, contract, contender	occur()
AnyParty (Patient)	IO	id, name, type, occasion, contract, contender	Withstands()
AnyParty (Physician)	IO	id, name, type, role, member	Follows()
AnyRules (MedicalLaw)	IO	name, id, list, description, validThrough	influences(), followedBy()
AnyAgreement (Decision)	IO	name, type, description, expiryDate, signingAuthority	influencedBy()
AnyOutcome (Treatment)	IO	id, name, type, description, implication	givenTo(), leadBy()
AnyMechanism (Medication)	IO	name, id, description, mode, method	leadsTo(), withstands()

AnyVerification (License)	IO	type, name, description, time, verificationEfforts	allows(), gives()
AnyReason (Self-Care)	IO	id, type, name, description, applicableDomain	helpedBy()
AnyType (Prescription)	IO	name, properties, subtype, department, id	offers(), allowedBy()
AnyMedia (InPerson)	IO	id, type, name, mode, method	offeredBy(), buys()
AnyLog (Records)	IO	id, name, type, description, date, logList	storedIn()
AnyEntity(Medicines)	IO	name, id, type, description, price	help(), boughtBy()

- During compliance AnyParty or AnyActor() complies with another AnyParty or AnyActor by using AnyMechanism(). (Who comply? What mechanism be used in compliance?) A patient complies with the physician's advice. The patient treats himself through prescribed medication.
- AnyParty() can frame AnyRule() during compliance. (How strictly are the rules followed? What happens when these rules are broken?) The physician follows the laws for a medical practice.
- AnyActor() initiates an agreement during the compliance process, (Who initiates the agreement in compliance process, and why?) The physician initiates the agreement to follow a prescription.

- AnyRule() updates AnyAgreement(). (What updates AnyAgreement?) (How is AnyAgreement() updated?) The laws governing a medical practice regulates whether the doctor give the patient an electronic presciption or a paper one.
- AnyAgreement() follows AnyMechanism(). (Who follows any AnyAgreement()?) The patient accepts the medication given by physician.
- AnyOutcome concludes AnyMechanism(). (How does AnyMechanism conclude?) Taking medication leads to the patient's recovery.
- AnyMechanism() follows AnyAgreement(). (What follows AnyAgreement()?) The physician follows the prescription regulations when deciding how often to have the patient take the medication.
- AnyVerification() identifies flaws. (Who identifies flaws in the process of compliance?) A person with medical license is allowed to practice medicine.
- AnyType() categorizes compliance. (What categorizes compliance?) The prescription is offered in person to the patient.
- AnyMedia() broadcasts AnyEntity or AnyEvent() to AnyActor/Party() through AnyMechanism(), (What does AnyMedia() connect through AnyMechanism()?) The written prescription connects the patient to his medicine.
- AnyLog() searches information about AnyEntity or AnyEvent(). (Where can a doctor find information about AnyEntity?) Information about previous medications is stored in the physician's record.
- AnyEntity() has a status. (What status does AnyEntity() have?) Medicines have an expiration date.

- AnyEvent() occurs with the involvement of AnyParty/Actor(). (How does AnyEvent() occur?) A patient may visit a physician for treatment.

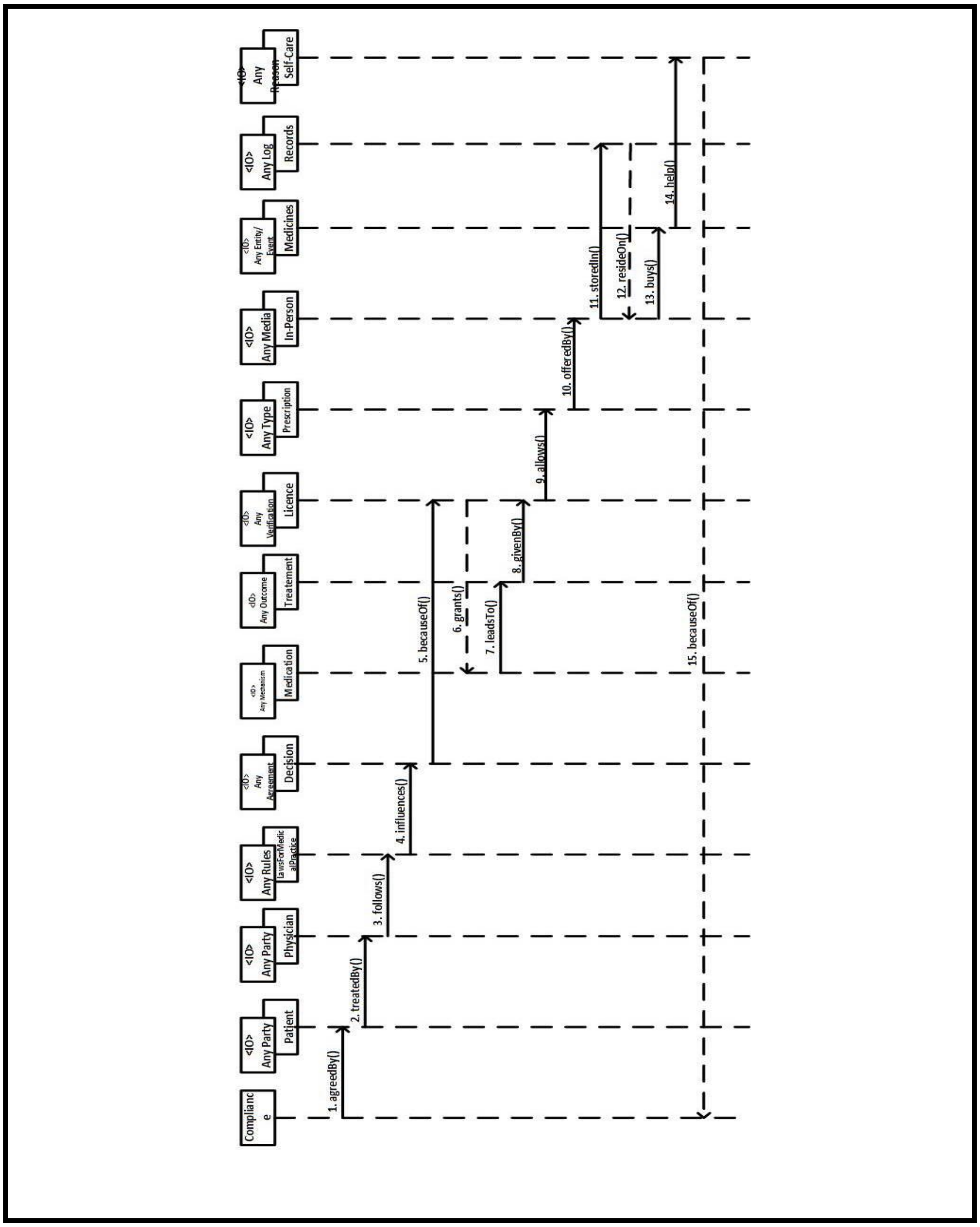
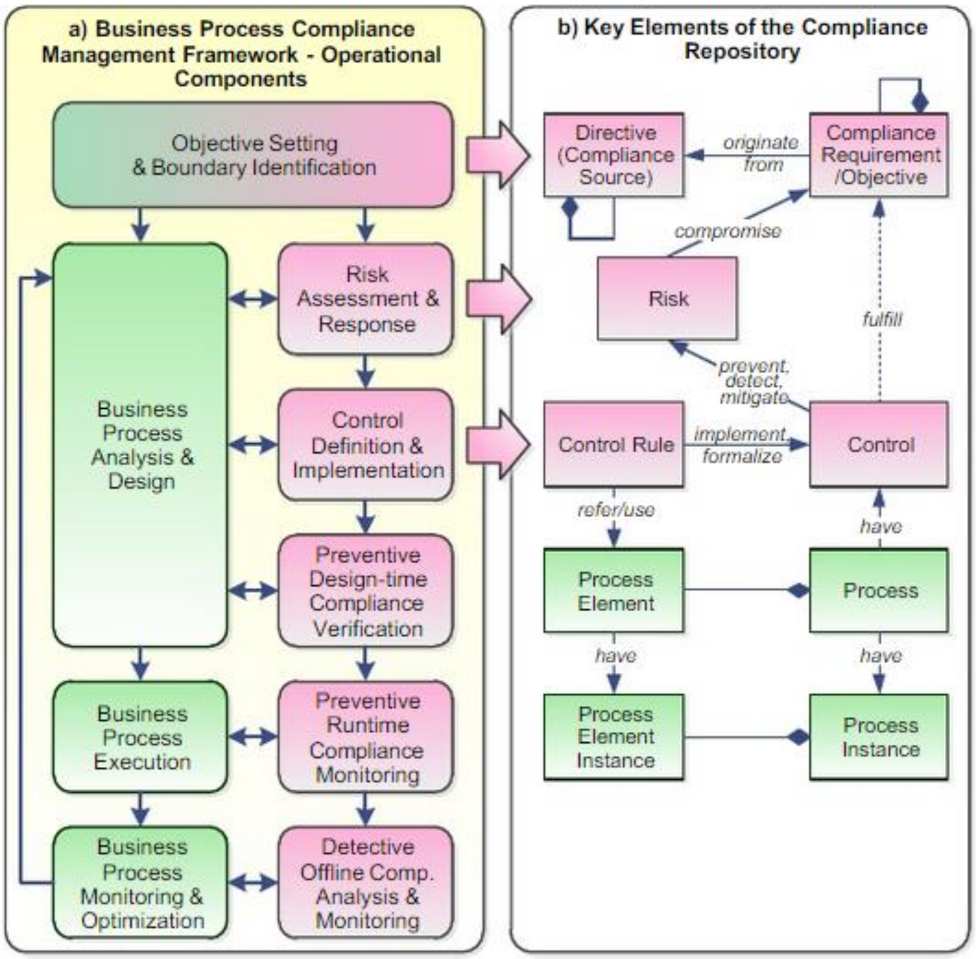


Figure 16. Sequence diagram for complying with a physician's prescription.

Related Pattern and Measurability

Related pattern: Traditional model. In this section, we compare the traditional model of compliance (Turetken, Elgammal, van den Heuvel, & Papazoglou, 2012, p.12). with the stable model using the compliance modifiers.



a) Business Process Compliance Management Framework - Operational Components
Objective Setting & Boundary Identification
Business Process Analysis & Design
Risk Assessment & Response
Control Definition & Implementation
Preventive Design-time Compliance Verification
Business Process Execution
Preventive Runtime Compliance Monitoring
Business Process Monitoring & Optimization
Detective Offline Comp. Analysis & Monitoring
b) Key Elements of the Compliance Repository
Directive (Compliance Source)
originate from
Compliance Requirement /Objective
compromise
Risk
fulfill
prevent, detect, mitigate
Control Rule
implement, formalize
Control
refer/use
have
Process Element
Process
have
have
Process Element Instance
Process Instance

Figure 17. Traditional Model for Compliance Consisting of Operational and Key Elements.

The BPCM (Business Process Compliance Management) and the theoretical model form the core parts of compliance repository (Turetken, Elgammal, van den Heuvel, & Papazoglou, 2012, p. 7). The part of the figure on the right side presents the model for compliance.

Measurability

The quality of a project should always be of a high standard. It is important to sustain the flexibility factor in a model, allowing it to accommodate changes that might be made in the future. Reusability is another important factor because changes always occur. The two factors that make the stability model more effective are

- generalization, which is more effective because it focuses on concepts that last forever; and
- expressiveness, which allows the focus to be on the domain instead of the IOs.

The following formula can check the factor of reusability: $RF = C_u/T_c$. RF is the reusability factor; T_c is the total number of classes, and C_u is the number of classes reused. With the traditional model, RF = 0/9 = 0. This means the traditional model is less likely to be reusable because no class is reusable. With the SSM, RF = 14/14 = 1. This means everything can be reused.

As compared to the traditional model, the stability model contains fewer IOs. The stability model focuses mainly on EBTs and BOs, whereas the traditional model is completely dependent on IOs. The total operations formula calculates the qualitative measurability: $T_{OP} = C * O_{PC}$. Here, T_{OP} equals the total number of operations; C equals the total number of classes; and O_{PC} equals the operations per class.The number of classes and the operations define the simplicity of the project.

For the traditional model, we suppose five operations per class and therefore, T_{OP}= 9 * 5 = 45. With the SSM, we could suppose one, two, or three operations per class during the same scenario. T_{OP} = 14 * O_{PC} = 16, 17, or 18.

Increased numbers of operations with fewer numbers of classes make a model simpler and more stable. Considering the above calculations, it is clear that the stability model has less complexity. See Table 14 for a breakdown of the SSM and traditional model.

Table 14
Compliance Comparison Between the Traditional Pattern and the SSM.

Adequacy	Traditional Model	Stability Model	Weighted Score in %	
			Traditional Pattern	SAP
Simplicity	Visually simple but does not provide a complete solution.	Visually complex but uses EBTs, BOs, and IOs to make the system stable and complete.	1	8
Ability to Understand	The traditional model needs to be changed as the application changes, which make them less understandable.	Everything is examined and the model does not need to be changed with the application, which makes them more understandable.	2	8

Systematic	Single layers mean some system areas are missed.	Many layers mean that all areas of the system are addressed..	6	27
Complete	The model fails to connect all internal aspects of the project.	This model is designed using all the concepts so it is complete and stable.	3	18
Stability	Unstable because it keeps only one application in mind.	Stable because the core is built of stable EBTs and BOs so only the IOs need to be changed..	1	20
Visual	Visually the model is straighforward, but much documentations is required to understand it.	Visually the model is difficult to understand because of the layers but it does not require as much documentation.	4	8
	Total Score:		17%	89%

Note: SAP = stable analysis pattern.

Known usage. The compliance pattern is exceptionally generic. It is a relevant example for numerous everyday situations. Each party or actor consents to what is suitable in his or her appropriate areas. The different situations where the compliance pattern is known to be valuable are

- <u>Prisoners are not given an option and must comply with the guards and the rules of the prison, or they face severe consequences</u>.

- A student complies with the teacher’s instructions in a classroom. Even when students do not wish to comply (i.e. by listening to a lecture), they comply with the teacher because they do not want to fail.
- When a superstar recommends a product, the fans have the option to comply with him or her by buying those products because their favorite celebrity is promoting them.

Summary. This chapter shows how the core knowledge of compliance can be used for numerous applications. It interprets SAP for compliance that is rugged, robust, and extendable over a wide number of applications. This model can be extended to identical contexts. Reusability of the pattern itself is a very big contribution. The pattern has been designed so that it be generic enough to handle numerous types of applications extending from it. Although the design was tested using numerous small, medium, and large applications, it still needs to be tested on more massive applications. However, based upon its current performance, we expect it to do well.

Advice SDP

The AnyAdvice SDP. Advice is a BO with the EBT of Advising. Advice is a term used to signify suggestions, opinions, and recommendations about certain situations in different contexts. Other names for advice are consultation, guidance, and instruction (“Advice,” 2016). The reason we chose this term for this pattern is because it applies to giving guidelines in any context or taking guidelines in any context. The generality of the term lead to an SDP for advice, and it also helps to categorize BOs.

Context. Advice is a recommendation about what should be done when addressing a problem, which decision to make, or how to manage a situation. Although advice is a suggestion given to a person or to a group, the impact can either be positive or negative. It is often a type of guidance concerning future actions, typically given by someone who is knowledgeable. Some examples are below.

Consider a scenario where a student (AnyParty) is looking for jobs at a software company (AnyParty) after his graduation. The student attends a career workshop (Advising) to obtain some guidance on the job application (AnyReason) process. Such workshops are called career workshops (AnyType). Students also attend job fairs (AnyEvent) in order to interact with their potential employers and receive suggestions (AnyAdvice) from the hiring managers. A student may use a resume as his marketing tool to apply for the jobs through online job portals (AnyMedia), which connect the employers to the profiles (AnyLog) of their potential employees.

Consider a mobile app (AnyMedia) which lets married couples (AnyParty) consult (AnyAdvice) other users of the app as if they were relationship experts (AnyParty). They do this when they feel they are not happy (AnyReason) because of problems in their relationship. Married couples might also seek counseling (Advising) at a counseling centers to avoid getting divorced (AnyCriteria). Divorce (AnyEvent) causes difficulties in the lives of the couple as well as their children. The app would connect such unhappy wives and husbands to relationship (AnyType) experts so that they can discuss their situation and repair their relationship. The app keeps the account (AnyLog) information of the users anonymous so that no user be reluctant to approach any expert.

Problem. The existing problem proves that there is no SDP that exists for AnyAdvice, which defined the core knowledge of advice. Once we have an SDP for AnyAdvice with Advising as its EBT, one can build as many applications as one wants using it. They the pattern is reusable without changing the core.

Functional requirements.

- Advising is recommending the best course of action to someone to address AnyType of problem. For example, it is extremely difficult to differentiate between good and bad products with all the products on the market. An individual or a group needs advising when choosing products. Advising is the EBT of the AnyAdvice SDP. It has attributes such as input, type, and fee. It has operations such as guide(), direct(), instruct(), and charge().
- AnyParty generally gives AnyAdvice as a suggestion or recommendation to someone to help them find a solution, to make a decision, or to manage a situation. AnyParty has a name, contact information, and at least two roles. An employee of an organization participates() in company projects, can playRoles() on the team by collectingData(), and can interact() with other employees.
- AnyAdvice applies to AnyEntity. For example, when romance is the theme for a novel, it may get advice on developing the male and female roles. If it is a thriller, then advice may be given on developing suspense. AnyEntity could be a product. AnyAdvice is given to AnyActor or AnyParty when purchasing an item. AnyEntity has a name, id, and type. If AnyEntityis a machine, it is expected to performFunction(). Two small machines may be a part of a bigger machine and

have a relationship() among them. AnyEntity has a status() depending on its operation and can be classified as new() or old().

- Advising is done for AnyEvent. It has attributes such as name, occasion, type, and outcome. Its operations can be appear(), happen(), and arrange().
- AnyCriteria is the foundation for AnyAdvice. It can be based on solving a problem, on forming a strategy, or on solving a problem. AnyCriteria are influenced by AnyReasonIt has attributes such as description(), describes(), and decides(). AnyCriteria has a description(), because whenever one gives AnyCriteria, the details of the criteria have to be mentioned for good understanding. Based on the criteria, one can decide() and give conclusions.
- AnyType further defines AnyReason. AnyEntity may have AnyType, too. AnyType has a name, an id, a property, and possibly a subtype. It has operations such as change(), operateOn(), subtype(), and classify().
- AnyLog records AnyAdvice given on AnyMedia in the form of comments. AnyLog is specific to the media. AnyLog allows its users to search for a specific record. It has attributes such as id, name, numberOfEntries, and reference. AnyLog operators include stored(), search(),insert(), or extend().
- AnyMedia is the method through which AnyAdvice is given. AnyMedia may be a phone call, a letter, or an email. The data and information about AnyAdvice is stored in AnyLog. AnyMedia has a name, an id, and a category. AnyMedia connects() different entities and events to each other. It allows AnyAdvice to be

logged and gives access() to advice. AnyMedia helps to broadcast() AnyAdvice and display() it as well.

- **Any Advice:** AnyAdvice is the view or thought of AnyParty. AnyAdvice can be given to a country, a party, or an individual. AnyType of AnyAdvice can be given.
- AnyReason is influenced by conditions and requirements. AnyAdvice has AnyReason, and AnyReason can be AnyType. It has operators such as description(), describes(), and decides().

Non-functional requirements.

- AnyAdvice should be appropriate within the context of the situation. An engineering student would not go to a professor in the school of business for advice on which engineering courses he should take. Preferably, AnyAdvice should be given by someone who has some prior knowledge or experience with the topic of the advice. An advisor should also use an appropriate tone and give the advice at an appropriate level.
- AnyAdvice should encourage AnyParty receiving it to follow the advice. If the advisor is discouraging or unenthusiastic, the person listening to the advice may not follow it.
- AnyAdvice that does not give a new perspective or inform AnyParty being advised is not helpful. Giving advice that someone already has heard or that someone already knows, does not help the person to make a decision. For example, if there is a dispute between a landlord and a tenant regarding broken

pipes, then the tenant could go to a lawyer to seek legal advice. The lawyer could provide advice about filing a court case against landlord along with guidance about terms and legal charges that could be brought against the landlord. The lawyer can provide the tenant with advice that is informative and helpful.

- In certain instances, giving AnyAdvice can change a person's actions in an effective way. When a doctor gives advice to his or her patient about the diet that he or she needs to follow, it depends on the patient to utilize or ignore that advice. If the patient refuses to follow the doctor's advice, it may cause problems or negative health results. However, if the patient follows the doctor's advice properly, it may result in better health. Here, the doctor's advice affects the health of the patient positively because it is effective.

Challenges and Constraints

Challenges. Consider the scenario where lawyer (AnyParty) charges a certain fee for recommendations (AnyAdvice) made to the clients (AnyParty). If the fee is too high, then the lawyer become inaccessible to the poor. There should be a balance between the advice being given and cost associated with it. A plan should be in place so all people have access to good advice.

Or, if a student (AnyParty) wants to enroll in a course (AnyEntity), he should go to his advisor (AnyParty) at the school to help him decide if the course is beneficial. Although the advisor can explain the details of a specific course, or the student might even ask his friends (AnyParty) about the coursework. If the advisor and friends have different opinions (AnyAdvice) about the course, then the student be confused. It is important for the student to weigh the advice he receives. The advice from his advisor should have more weight than the advice from his friends.

In another case, there are websites that give advice on how to improve a relationship after a user (AnyParty) posts his or her problem (AnyReason) on the forum page. Other users (AnyParty) of the website may comment (AnyAdvice) on the post and offer advice. The advice they offer could be incorrect. The other users may or may not be experts. One should not follow any advice blindly because not every advice is correct. AnyParty should check if the one offering advice is knowledgable or not.

When an actress (AnyParty) goes for a haircut, she does not go to her friends. Instead, she goes to a professional beautician (AnyParty) to have her hair cut in a way that suits her style and looks (AnyReason). Although a friend might be able to offer a nice haircut, it would not be of the same caliber as a professional. When seeking AnyAdvice, it is important to ask the most professional person for advice. In the case of an actress, her hairstyle would affect the way her fans react to her. A non-professional hair stylist may not be able to give good (AnyType) advice (AnyAdvice) to the actress. Therefore, the person should always choose an expert of a domain for which he or she needs advice. Table 15 summarizes these scenarios.

Table 15
The Challenges and Constraints of Sample AnyAdvice Scenarios

Title	Problem	Constraint	Solution
Advising Costs	A lawyer's fee is too high for some clients.	Lack of resources	Lawyers may need to adjust their fees based on their clients.
Academic	When different people	Opposing	The student should

advising	have opposing recommendations, the advise is difficult to follow.	advice from different people	prioritize which person's advice receives the most weight.
Relationship advice	When a person seeks help online for issues such as relationships, the people providing the advice may not be professionals.	Following advice blindly.	It is important to weigh all advice before following it.
Advice from a professional	An actress has her hair done by a professional.	Choosing an advisore can be difficult	A person should always choose the most professional person from whom to seek advice.

Constraints.

- Advising has to be achieved through AnyParty as shown in the class diagram for AnyAdvice SDP (Figure 18); advising cannot be achieved without AnyAdvice.
- One or more AnyCriteria guides AnyAdvice.
- AnyParty defines AnyCriteria and performs Advising.
- AnyAdvice has AnyReason.
- AnyReason names AnyType.
- AnyEntity or AnyEvent determines AnyType.
- AnyLog resides on AnyMedia.
- AnyEntity or AnyEvent is listed on AnyMedia.

Solution. We provide core knowledge by advocating AnyAdvice SAP and utilizing SSM concepts. The given solution provides an advice pattern that can be used to build numerous applications.

Pattern structure. The relation between the EBTs and BOs within the Any Advice SDP is shown in Figure 18.

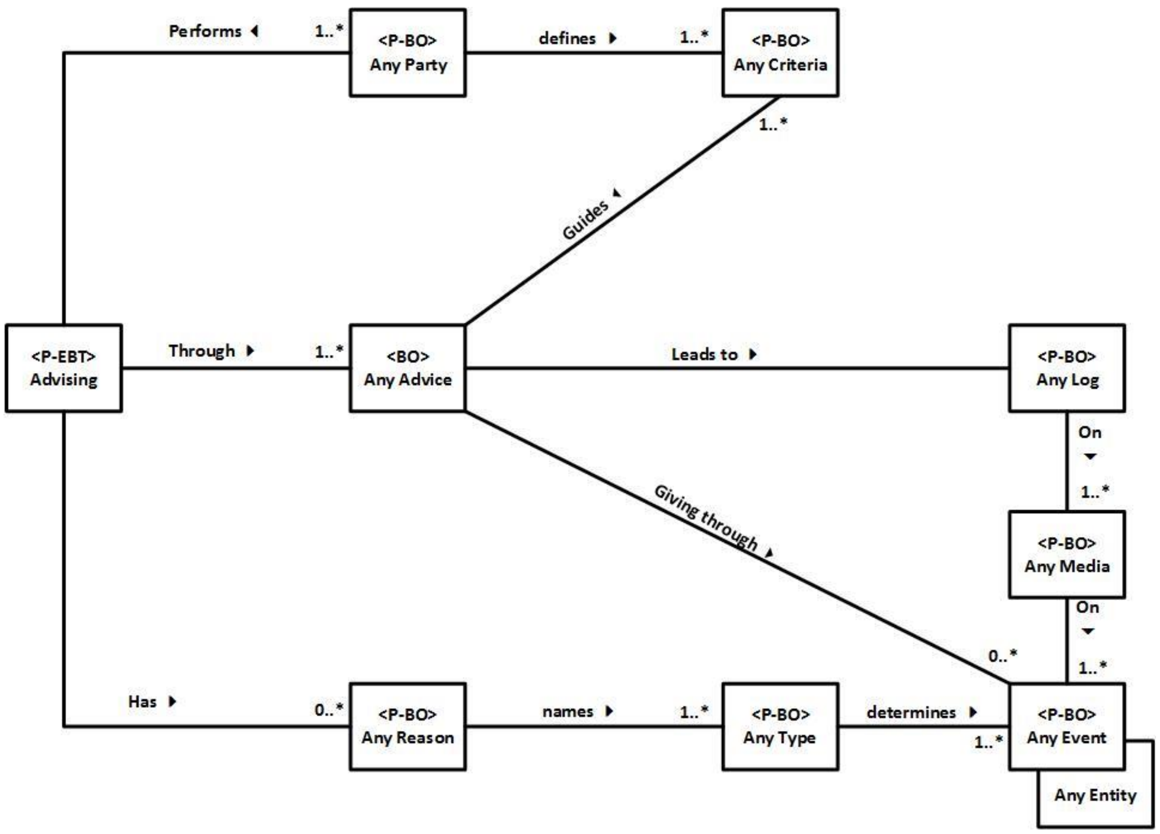


Figure 18. The AnyAdvice SDP.

Pattern participants. The participants within the pattern are the EBT and BOs. We describe them as classes and patterns:

Classes. AnyAdvice represents the original advice process. The core of this SDP is presented here. Various attributes and operations conduct the advising process.

Patterns.

- Advising represents an EBT. It offers recommendations to AnyParty.

- AnyParty represents the party that provides or seeks advice. It models every party involved in the advising process. A party could be a person, organization, country, or political party.
- AnyReason represents the reason advice is needed or provided. It could be a fact, a situation, an explanation, a rationale, or a condition.
- AnyType represents different types of advice that can be provided to or that are needed by a party. Some types are legal and academic advice.
- AnyEven**t** represents the event requiring advice.
- AnyEntity represents the entity requiring advice.
- Any Criteria represents the standard according to which anything could be chosen. For example, there could be safety criteria in an airplane that should not be violated by the passengers.
- AnyLog represents the records or the place where data and files are stored. AnyLog resides on AnyMedia.
- AnyMedia represents the medium through which advice occurs. For instance, one can advise about a holiday location over the phone or in person. Some others would like to receive such advice via email.

Class diagram description.

- AnyParty performs Advising.
- AnyAdvice is the Advising.
- Advising has AnyReason.
- AnyAdvice is recorded on AnyLog.

- AnyLog is stored on AnyMedia.
- AnyEvent is promoted on AnyMedia.
- AnyCriteria is defined by AnyParty that guides AnyAdvice.
- AnyAdvice is given through AnyEvent.
- AnyEntity is within AnyEvent.
- AnyReason names AnyType.
- AnyType determines AnyEvent.

CRC Cards.

Table 16
CRC Card Information for the Advice SAP

Participant	Duty	Attributes	Collaboration	
			Client	Server
Advice(EBT)	Provide advice	domain, name, context, id, category	AnyParty**,** AnyAdvice, AnyReason	giveAdvice(), encourage(), seekOpinion()
AnyParty (BO)	Seeking advice	name, id, type, address, phoneNumber, email	AnyCriteria, Advice	pursueHelp(), giveEncouragement(), stopAssistance()
AnyAdvice (BO)	To offer advice	id, status, application, mumberOfrecommendations, description, context	AnyCriteria, Advising, AnyLog, AnyEntity, AnyEvent	offerInformation(), recommend(), leadsTo(). suggest()
AnyCriteria (BO)	Enfoce limitation	id, type, severity, name, description	AnyParty, AnyAdvice,	exerciseRestriction(), control(), obstruct()

AnyReason (BO)	Specify details	id, description, proof, justificaiton, context	Advising AnyType	conclude(), examine(), resolve()
AnyType (BO)	Designate category	id, property, interfaceList, methodList, subtype, typeName	AnyReason, AnyEntity, AnyEvent	classify(), attach(), detach()
AnyMedia (BO)	To display	id, name, type, capability, entry, securityLevel	AnyEntity, AnyEvent, AnyLog	display(), store(), capture(), broadcase()
AnyLog (BO)	To store records	id, name, number, size, location, references	AnyMedia, AnyAdvice	search(), insert(), extend(),
AnyEntity (BO)	Exist independ-antly	entityName, id, type, position, state, status, isAlive	AnyType, AnyMedia, AnyAdvice	performFunction(), relationship(), new()
AnyEvent (BO)	To happen or occur	name, id, type, occasion, contract, contender	AnyType, AnyMedia, AnyAdvice	compete(), celebrationFor(), rank()

Consequences. The model is planned to provide flexibility and reusability. It can serve as a reusable segment inside many applications including the application of advice. The BOs and the patterns are sufficiently non-exclusive to be suitable for reuse in any application.

The idea of advice is exceptionally dynamic, and it takes diverse structures in light of the setting in which it is connected. Subsequently, it is imperative to supplement the current model with other domain-particular ideas before it can be incorporated into the framework. This is a result of the conceptual aspect of the idea and the model being exceptionally nonexclusive. It is not so much an impediment to the framework, yet is an exchange with a specific end goal to fulfill more applications and domains. The model also has

- understandability: The AnyAdvice design pattern exhibits the idea of giving proposals and suggestions; it uses AnyAdvice in an effortlessly understandable design through the Advising EBT;
- flexibility: The AnyAdvice design pattern is sufficiently not specific to a single domain; this is delineated through basic interfaces to permit access only to chosen methods for any of the BOs; and
- extensibility: The pattern can be extended by connecting particular classes like BOs into the pattern; consequently, the framework gives an enormous state of extensibility to suit applications in different areas.

Applicability.

Table 17
Applicability of the Advising EBT Across Several Disciplines

BO	Personal Advice	Group Advice	Company Advice	Legal Advice	Financial Advice
AnyParty	Friend, wife	Online forum, user community	Apple	Lawyer, client, court	Investor, client
AnyAdvice	Good choice	Think twice	Needs improv	Litigation	No profit

AnyReason	Difficult to choose	Value for money	Bad product release	Lawsuit against restaurant	Buying stocks
AnyEntity/ AnyEvent	Toy	Laptop	Software	Buffet	Share
AnyType	All look good	Function	Perform-ance	Poor food	Stocks
AnyCriteria	Brand and make	Quality	Testing	Bill	Maturity
AnyMedia	In person, email, phone	Online, focus group, email	Online portal	In person, phone call, email	Phone, email, in person,
AnyLog	Message history, chat history	List of comments	Company records`	Email history	Chat history, email history

Case Study 1: Advice About Buying a Toy From Walmart

Consider a scenario where a customer (AnyParty) visits a Walmart (AnyParty) online store to purchase (AnyCriteria) a toy (AnyEntity) for her son's satisfaction (AnyReason). The customer needs advice on the offered toys. So, she checks an online forum (AnyLog) for the online store (AnyMedia) and decides to follow the advice (AnyType) given by several users about the toys. The buyer reads comments posted about the toy in the forum and follows the advice (AnyAdvice) given by people. Finally, she buys a toy after making the comparison. Figure 19 provides the class diagram of this case study.

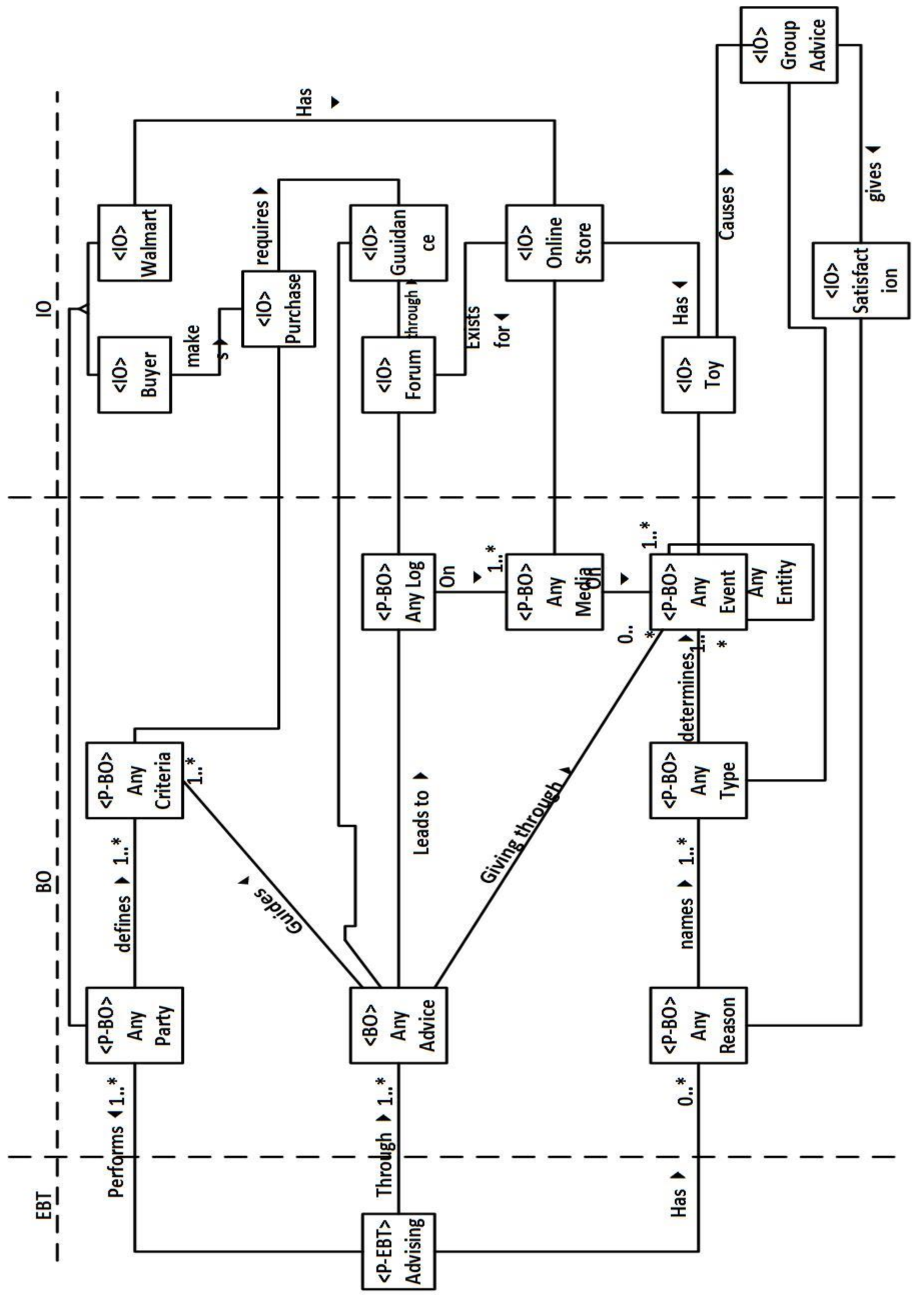


Figure 19. Purchasing a toy from Walmart. A class diagram showing the selling EBT, its BOs, and its IOs.

Use case and description. Consider the use case for buying a toy at Walmart in Table 18.

- Use Case ID: 001
- Use Case Name: Advice on Buying a Toy From Walmart

Table 18
Use Case 1 for AnyAdvice

Actor	Role
AnyParty	Customer, reviewers

Class	Type	Attributes	Operations
Advising	EBT	domain, name, context, targetAudience, advisingTone	giveAdvice()
AnyParty()	BO	name, type, occupation, gender, age	seekOpinion()
AnyCriteria()	BO	name, description, type, purpose, constraints	control()
AnyAdvice()	BO	name, status, description, isGiven, isRecieved	offerInformation()
AnyReason()	BO	description, level, summary, name, constraints	conclude()
AnyEntity	BO	name, description, type, price, barCode	performFunction()
AnyEvent	BO	name, id, type, occasion, contender	occur()
AnyType()	BO	id, name, category, subtype, quantity	classify()
AnyMedia()	BO	id, name, type, entry, data	display()
AnyLog()	BO	id, name, refernece, date,	search()

		size, location, type	
AnyParty(Walmart)	IO	name, type, occupation, gender, age	sell()
AnyParty(Buyer)	IO	name, type, occupation, gender, age,	visits()
AnyCriteria(Purchase)	IO	mame, description, type, purpose, constraints	resultOf()
AnyAdvice(Guidance)	IO	name, status, description, isGiven, isRecieved	generate()
AnyReason(Satisfaction)	IO	description, level, summary, name, constraints	ensures()
AnyEntity(Toy)	IO	name, description, type, price, barCode	orderFrom()
AnyType(Group Advice)	IO	id, name, category, subtype, quantity	provide()
AnyMedia(Online Store)	IO	id, name, type, entry, data	contain()
AnyLog(Web Forum)	IO	id, name, reference, date, size, location, type	giveAdvice()

- Advising() is for giving advice to AnyParty(), (Who gives advice to AnyParty()?) Advice is given to a buyer.
- AnyParty() seeks the opinion of another party, (Why does AnyParty() seek an opinion?) The buyer seeks an opinion from the salesperson.
- AnyCriteria() controls AnyAdvice(). (Why does AnyCriteria control AnyAdvice()?) Guidance controls a purchase.
- AnyReason() concludes advising. (What concludes advising?) Satisfaction concludes a purchase.

- AnyEntity() performs a function during advising. (What function does AnyEntity() perform?) A toy performs the function of amusing.
- AnyEvent() occurs during advising. (What occurs during advising?*)* Meeting a professor during advising sessions is an event.
- AnyType() classifies AnyAdvice(). (What classifies AnyAdvice()?) Group advice is advice given to a group instead of an individual.
- AnyMedia() displays the results of AnyAdvice(). (What is displayed by AnyMedia()?) An online store the displays toys for sale.
- AnyLog() searches for information about AnyEntity or AnyEvent. (What is searched by AnyLog()?) A web forum is used to search for reviews about a toy.

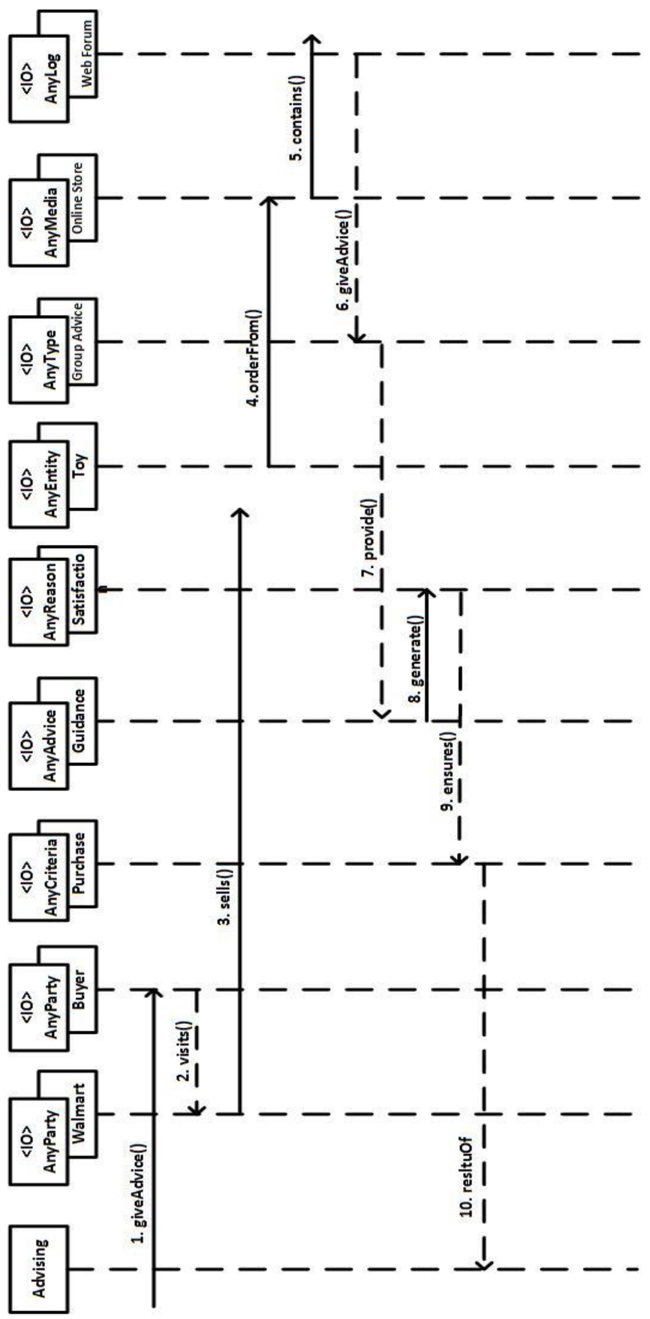


Figure 20. Sequence Diagram for AnyAdvice SDP

Case Study 2

Scenario. Consider a scenario where a user (AnyParty) visits the Apple (AnyParty) online portal (AnyMedia) to perform testing (AnyCriteria) in order to improve the performance (AnyType) of software (AnyEntity). Apple designated this activity to external users for quick product release (AnyReason) with high-performance testing. But, the user needs the proper advice (AnyAdvice) from Apple technicians to complete improving the software modules or components. The Apple technicians redirect the user to company records (AnyLog) related to the software so the user can perform beta testing. As a result, the user refers to the company records, performs beta testing, and helps Apple release the product on time. Figure 21 provides the class diagram of this case study.

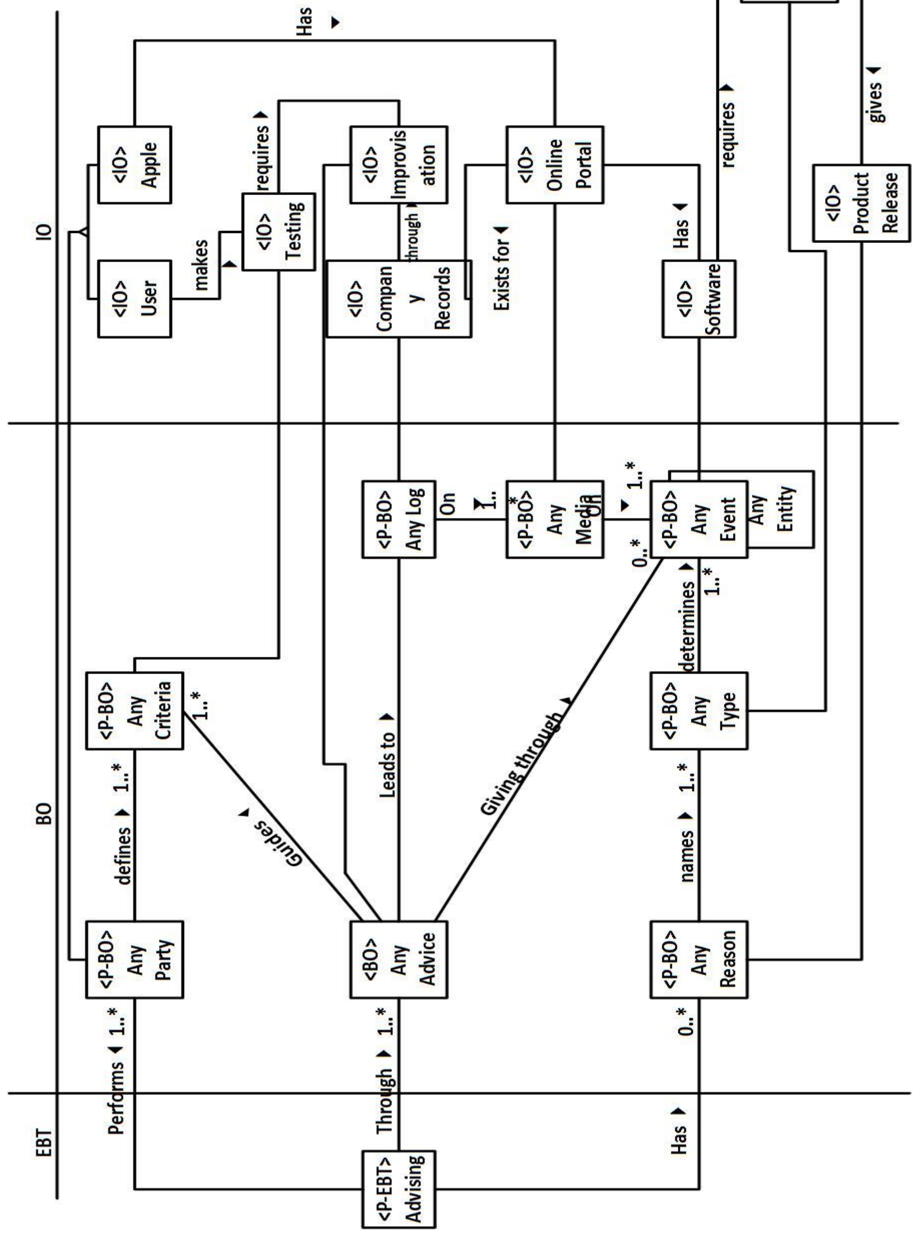


Figure 21. The class diagram for an Apple user seeking advice. showing the EBT, BOs, and IOs.

Use case and description. Consider the use case for obtaining software from Apple in Table 19.

- Use Case ID: 002
- Use Case Name: Obtaining Software from Apple

Table 19
Use Case 2 for Advising: Obtaining Software from Apple

Actor	Role
AnyParty	Apple, user

Class	Type	Attributes	Operation
Advising	EBT	department, advisorName, domain, title, context	encourage()
AnyParty()	BO	name, id, type, age, gender	assist()
AnyCriteria()	BO	title, description, type, purpose, constraint	obstruct()
AnyAdvice()	BO	id, description, status, isGiven, isReceived	suggest()
AnyReason()	BO	description, level, summary, name, constraints	resolve()
AnyEntity()	BO	name, description, type, price, versionNumber	operate()
AnyEvent()	BO	name, id, type, occasion, contender	rank()
AnyType()	BO	id, name,	include()

		category, subtype, quantity	
AnyMedia()	BO	id, name, type, entry, data	store()
AnyLog()	BO	id, name, reference, date, location, list, type	insert()
AnyParty(Apple Store)	IO	name, id, address, contactNumber, websiteUrl	seekForAdvice()
AnyParty(User)	IO	name, id, type, age, gender	assistInTesting()
AnyCriteria(Testing)	IO	title, description, type, purpose, constraint	check Modules(), checkComponents()
AnyAdvice(Improvisation)	IO	id, description, status, isGiven, isReceived	referToRecords()
AnyReason(ProductResealse)	IO	description, level, summary, name, constraints	preReleaseTesting()
AnyEntity(Software)	IO	name, description, type, price,, versionNumber	purchaseFrom()
AnyType(Performance)	IO	id, name, category, subtype, quantity	performanceMeasure()
AnyMedia(Online Portal)	IO	id, name type, entry, data	holds()
AnyLog(Company Records)	IO	id, name, reference, date, location, list, type	indicate()

- Advising() encourages AnyParty(), (Who gives advice to AnyParty()?) Advice is given to a user.

- AnyParty() assists another party, (Why does AnyParty() assist?) An Apple technician assists the Apple user during system testing.
- AnyCriteria() obstructs AnyAdvice(). (Why does AnyCriteria() obstruct AnyAdvice*()?)* While testing, advising is not available.
- AnyReason() resolves advising, (What resolves advising?) Testing precedes product release.
- AnyEntity() operates during advising. (What operation does AnyEntity() perform?) Apple online software used by a user for testing.
- Advising includes AnyType(). (What includes AnyType()?) Performance leads to improvement.
- AnyMedia() stores the results of AnyAdvice(), (What is stored by AnyMedia()?) Company records hold past records of online portal testing.
- AnyLog() indicates information about AnyEntity or AnyEvent. (What is indicated by AnyLog()?) Company records indicate performance history.

Related pattern and measurability.

Related pattern.

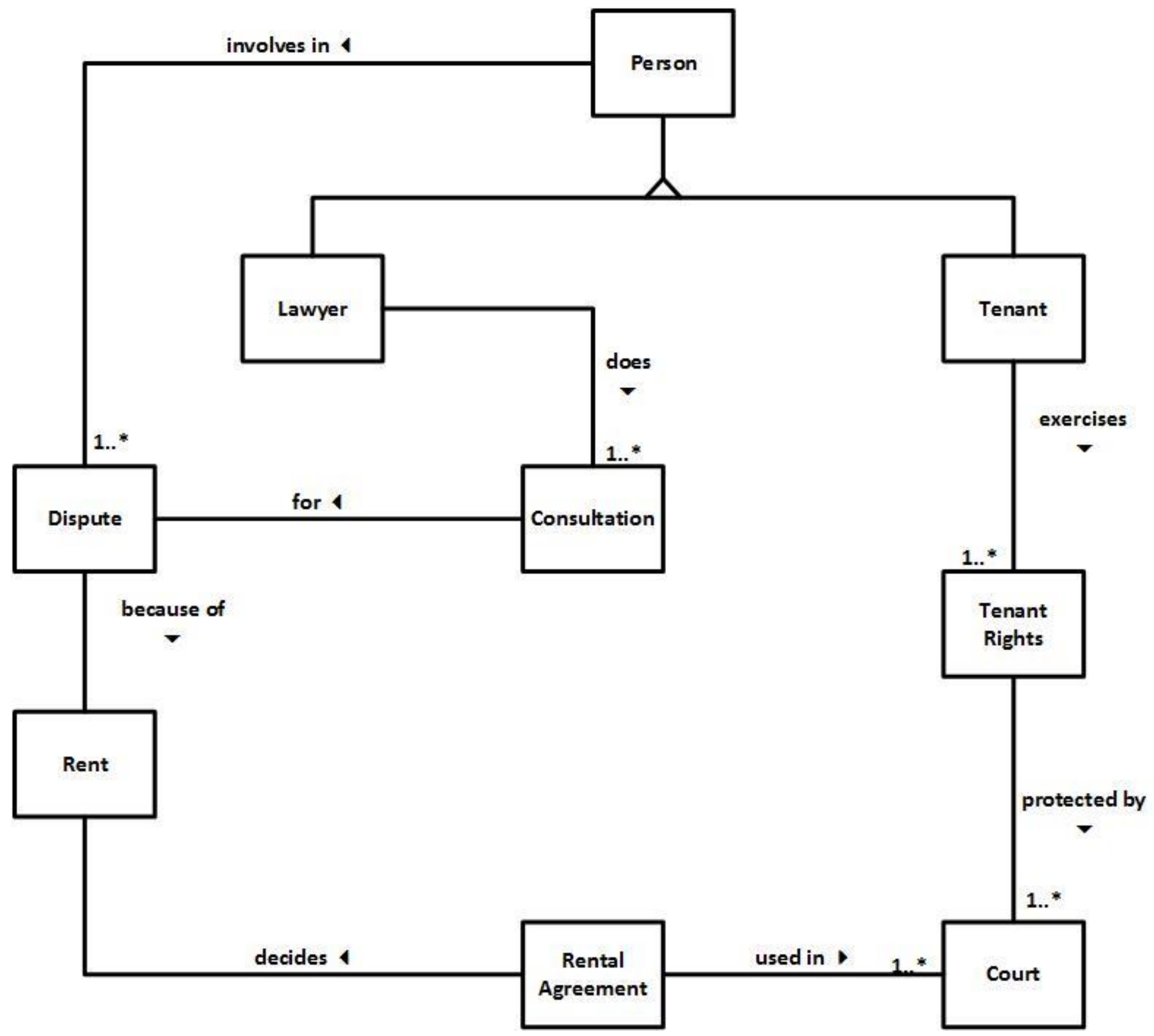


Figure 22. The traditional model for advice.

Comparative study using model adequacies.

Table 20
AnyAdvice Comparison Between the Traditional Pattern and the SAP

Adequacy	Traditional Model	Stability Model	Weighted Score in %	
			Traditional Pattern	SAP
Reusibility	Specific to certain domains and deal with only tangible objects or IOs. Not reusable.	Use EBTs and BOs as an unchangable core. Reusable.	3	30
Ability to Understand	The traditional model needs to be changed as the application changes, which make them less understandable.	Everything is examined and the model does not need to be changed with the application, which makes them more understandable.	1	15
Maintenance	Difficult and costly.	Simple and inexpensive.	2	15
Cost	Initially cheaper but costs balloon as changes are needed.	Initially more expensive and more time-consuming, but the costs are low when changes are needed.	1	9
Stability	Unstable because it is built on IOs.	Stable because the core is built of stable EBTs and BOs.	0	30
	Total Score:		7%	99%

Note: SAP = stable analysis pattern

Measurability. The quality of a project should always be of a high standard. It is important to sustain the flexibility factor in a model, allowing it to accommodate changes that might be made in the future. Reusability is another important factor because changes always occur. The two factors that make the stability model more effective are

- generalization, which is more effective because it focuses on concepts that last forever; and
- expressiveness, which allows the focus to be on the domain instead of the IOs.

The following formula can check the factor of reusability: $RF = C_u/T_c$. RF is the reusability factor; T_c is the total number of classes, and C_u is the number of classes reused. With the traditional model, RF = 0/9 = 0. This means the traditional model is less likely to be reusable because no class is reusable. With the SSM, RF = 10/10 = 1. This means everything can be reused.

As compared to the traditional model, the stability model contains fewer IOs. The stability model focuses mainly on EBTs and BOs, whereas the traditional model is completely dependent on IOs. The total operations formula calculates the qualitative measurability: $T_{OP} = C * O_{PC}$. Here, T_{OP} equals the total number of operations; C equals the total number of classes; and O_{PC} equals the operations per class.The number of classes and the operations define the simplicity of the project.

For the traditional model, we suppose five operations per class and therefore, T_{OP}= 9 * 5 = 45. With the SSM, we could suppose one, two, or three operations per class during the same scenario. $T_{OP} = 10 * O_{PC}$ = 11, 12, or 13.

Increased numbers of operations with fewer numbers of classes make a model simpler and more stable. Considering the above calculations, it is clear that the stability model has less complexity.

Summary. This thesis demonstrates how the core knowledge of AnyAdvice can be used in numerous applications. It also interprets an SDP for AnyAdvice that is easy to maintain, cost-effective, less time-consuming, robust, and flexible. This model can be extended for many identical contexts. Reusability of the pattern itself is a considerable contribution along with understandability, flexibility, and extensibility.

Chapter 4: Mid-size Documentation of Stable Analysis and Design Patterns

AnyCommitment SDP

Pattern name. AnyCommitment is a BO. Is a promise to do something. It is an agreement, assurance, guarantee, vow, or obligation to perform an action. Other words for commitment include pledge, promise, responsibility, and duty ("Commitment," 2016). Generality is the main reason for choosing this term as it is appropriate for all the possible scenarios of AnyCommitment. This leads to the SDP of AnyCommitment.

Context. AnyCommitment has applications in different domains such as business, education, medicine, and sports. Some types of commitment are consent AnyCommitment, declaratory AnyCommitment, default AnyCommitment, divine AnyCommitment, and investigative AnyCommitment. The following are some of the contexts where we can apply AnyCommitment as an SDP.

One of the primary contexts is financial commitment. Consider a scenario where parents (AnyParty) are obligated to pay the living expenses (AnyDeliverable) of their child (AnyParty) after a divorce (AnyEvent). This is a financial obligation (AnyCommitment) for the parents because they need to follow the state law (AnyCriteria). The parents pay monthly expenses through a bank check (AnyMedia), which is a legal (AnyType) method of payment.

Capital commitment is another important context. Capital commitment (AnyCommitment) is a promise made by an investor (AnyParty) to fund a venture capital (AnyParty) investment for its functioning. An investor buys an equity fund (AnyEntity) by agreeing to give money to the company. The investor has an expiration date

(AnyCriteria) on the contract. In the private equity market (AnyMedia) when one invests in the right businesses, the investor gains a huge reward (AnyDeliverable) in returns.

Problem. As of now, there is no SDP existing for AnyCommitment that defines the core knowledge of AnyCommitment. Once we have an SDP for AnyCommitment, then we can build as many applications as we want by using it repeatedly and eventually making the pattern reusable without changing the main core. The following are some of the functional and non-functional requirements.

Functional requirements.

- Obligation represents a situation where someone is bound to perform a task. It has attributes such as context, isWilling, and reason. It has operations such as oblige(), require(), and force().
- AnyCommitment represents the act of committing. AnyCommitment has attributes such as context, type, and level. This class has operations such as execute(), invest(), and deliver().
- AnyParty represents a party that is involved in the commitment. The party could be a country, a political party, an organization, or a person affiliated with an organization. AnyParty has attributes such as name, id, and location. It has operations such as participate(), collectData(), and interact()
- AnyCriteria represents something that prevents the satisfying of a commitment. It has attributes such as description, name, and id. The operations of AnyCriteria can be prevent(), test(), and measure().

- AnyDeliverable represents the thing that is expected to be delivered during AnyCommitment. It has attributes such as name, description, and status. It has operations such as bear(), convey(), distribute(), and transport().
- AnyType represents the type of commitment. AnyType has attributes such as id, name, interfaceList, methodList, and property. The operations are change(), categorize(), and subtype().
- AnyMedia represents the media though which AnyCommitment takes place. Data and information reside inside this medium. It has attributes such as name, availability, and isAvailable. It has operations such as display(), store(), capture(), broadcast(), and connect().
- AnyEntity is the object for which AnyCommitment is done and is named by AnyType of AnyCommitment. It has attributes such as id, name, and status. Its operations can be to update(), relationship(), and type().
- AnyEvent is the reason for which AnyCommitment is done and is named by AnyType of AnyCommitment. It has attributes such as name, occasion, type, and outcome. Its operations can be to appear(), happen(), and arrange().

Non-functional requirements.

- AnyCommitment needs to be relevant. Suppose a tenant complains about a bed bug problem in his apartment. If the landlord promises to treat the problem with a treatment for cockroaches, it not help and is completely irrelevant. The commitment needs to be a treatment for bed bugs and not one for cockroaches. In the same way, AnyCommitment should be relevant.

- AnyCommitment needs to be doable. Consider another situation where a builder promises to build an office building in a month. If the builder knows he cannot finish in a month, then the commitment should not be given in the first place. Every commitment made should be doable.
- AnyCommitment must be acceptable. In a commitment like marriage, both the bride and bridegroom must accept each other as life partners. If one does not want to marry the other, then there is a possibility that the marriage will end in divorce. Therefore, a commitment should always be acceptable.
- AnyCommitment made should be legal, and it should satisfy all the necessary legal requirements. For example, a person can make a legal commitment to a bank to pay back a house loan. However, the person is bound by the legal terms of the contract to repay the loan. If the person is required to make monthly installment payments and fails to do so, the bank can sue the person for the money and take away the house. If a commitment is not bound within the limits of the law, then that commitment is not valid within a legal context.

Table 21
Application for Commitment

EBT	BO	Financial	Capital	Marriage	Organi-zation	Trial
Obligation	AnyParty	Parent, Child	Investor, Venture capitalist	Bride, Groom	Employee, Employer	Accused, Judge

AnyCriteria	State Law	Total Years	Compatib ility	Layoffs	Crime Evidence
AnyCommi tment	Financial Obli-gation	Capital Commit-ment	Legal Bonding	Comple-tion of Project	Legit-imacy
AnyDeliver able	Liability	Remuner ation	Civil Stature	Task	Facts
AnyEntity	-	Equity Fund	-	Project	-
AnyEvent	Divorce	-	Marriage	-	Murder
AnyType	Legal	Invest-ment	Mono-gamy	Software	Legal
AnyMedia	Check	Equity Market	Matri-mony Website	Email	Court

Solution. This paper provides a solution by utilizing the SSM to extract the core knowledge of AnyCommitment. It focuses on how AnyCommitment should be given. The solution also provides an SDP that incorporates various methods to apply AnyCommitment and to build numerous applications on cross-platform domains. Figure 23 shows the class diagram of the AnyCommitment SDP.

The solution begins when AnyParty creates an obligation (Obligation) for AnyEntity or AnyEvent and identifies one or more criteria (AnyCriteria) for that obligation. AnyCommitment can inherently have one or more types (AnyType), which determine AnyEntity or AnyEvent. AnyCriteria influences AnyCommitment, which results in zero or more deliverables. AnyMedia then relies on AnyCommitment and AnyEntity or AnyEvent.

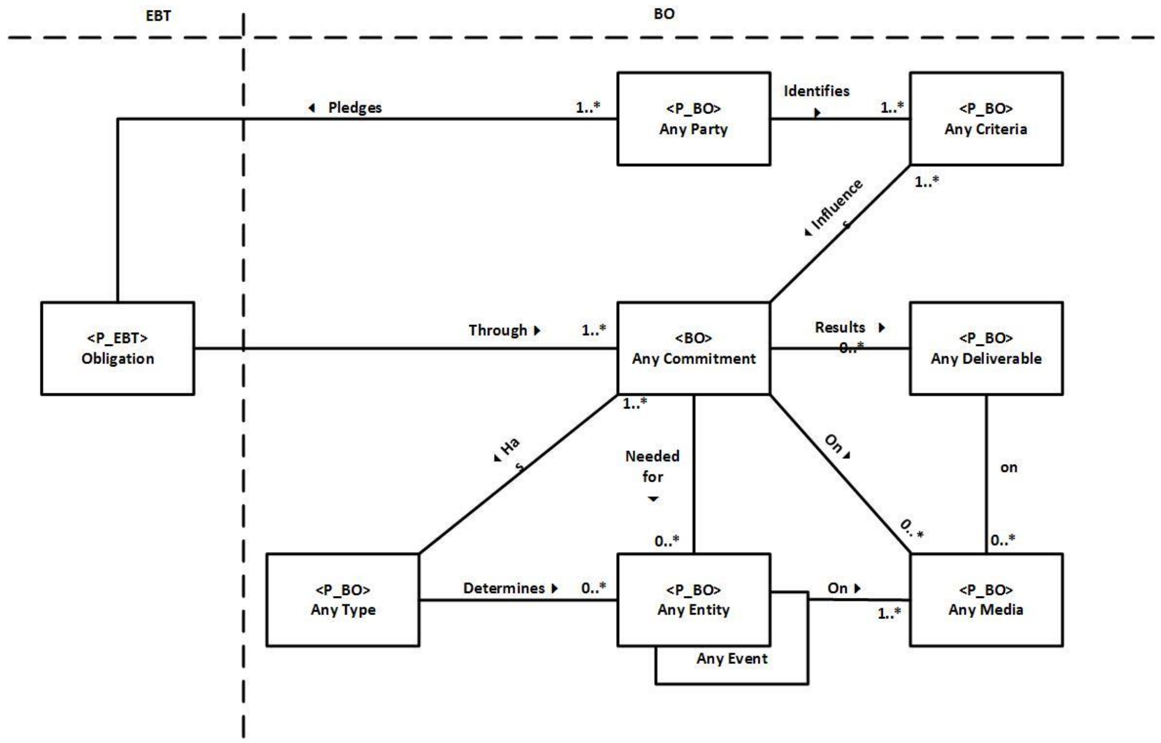


Figure 23. AnyCommitment SDP.

Example of financial commitment. Consider the scenario where parents are obligated to pay the living expenses of their child after a divorce. Figure 4.2 provides the class diagram of this case study.

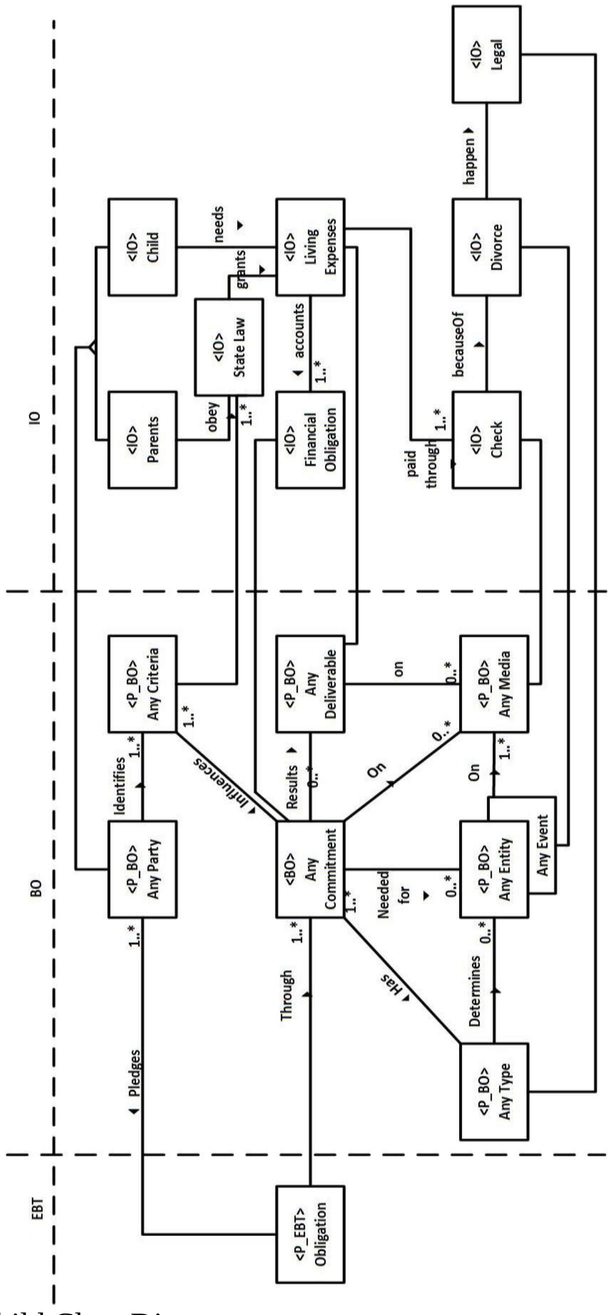


Figure 24. Parent-Child Class Diagram.

Use case and description. Consider the use case for obtaining software from Apple in Table 22.

- Use Case ID: 001
- Use Case Name: Obtaining Software From Apple

Table 22
Use Case 1 for AnyCommitment

Actor	Role
AnyParty	Parent, child

Class	Type	Attributes	Operations
Obligation()	EBT	description, name, context domain, state	require()
AnyCommitment()	BO	context, type, level, name, status	assure()
AnyParty()	BO	name, id, type, phoneNumber, age, address, gender	participate()
AnyDeliverable()	BO	id, description, status, name, statement, date	output()
AnyCriteria()	BO	name, description, anyProof, criteriaLimit, caseNo, anyConstraints	influence()
AnyEntity()	BO	name, id, type, position, state, status, isAlive	names()

AnyEvent()	BO	name, id, type, occasion, contract, contender, frequency	happen()
AnyType()	BO	id, property, subtype, name, interfaceList, methodList	classify()
AnyMedia()	BO	name, type, category, description, usedFor, location	resideOn()
AnyParty(Parent)	IO	dateOfMarriage, name, address, id, email, phoneNumber	obeyOrders()
AnyParty(Child)	IO	dateOfBirth, name, address, height, hairColor	recieveCompensation()
AnyDeliverable(Living Expenses)	IO	id, description, status, name, statement, deliveryDate	paidThroughCheck()
AnyCriteria(State Law)	IO	name, anyProof, description, criteriaLimit, caseNo, severity, anyConstraints, threshold	followRules()
AnyEvent(Divorce)	IO	name, id, type, occasion, frequency, objective, policy	takeDivorceDecision()
AnyType(Legal)	IO	id, property, subtype, interfaceList, methodList, type	legalCompensation()
AnyMedia(Check)	IO	name, type,	sendTo()

	category, description, usedFor, location

- Obligation requires AnyCommitment. The technical components of the obligation include the answers to the following questions. Who requires AnyCommitment? What is the burden of the Obligation? Who takes the accountability for an Obligation? In the case study, state law requires a commitment from parents and every parent needs to follow the state law.
- AnyParty participates in AnyCommitment. For this requirement, consider the answers to the following. Who participates in AnyCommitment? What does AnyCommitment guarantee? In the case above, the parents have the financial obligation.
- AnyCommitment assures something. We need to know what AnyCommitment assures. What is assured during AnyCommitment? Financial obligations are a part of living expenses, which means parents have to pay for the financial obligations of their child.
- AnyDeliverable is an output of AnyCommitment. The technical components of AnyDeliverable include the answers to the following questions. What is the output of AnyCommitment? Is the AnyDeliverable realistic? What need does AnyDeliverable fulfill? In the example, AnyDeliverable is a check or any other mode of payment.

- AnyEvent happens during AnyCommitment. To define this, we need to know what exactly happens during AnyCommitment? What type of AnyEvent occurs? What is the result of AnyEvent? When the parents decide to separate in the example, it is through a legal divorce.
- AnyType classifies AnyCommitment. Consider answers to the following questions. What classifies AnyCommitment? What categories of AnyType are necessary? Because the government dictates state law, using state law is a legal approach.
- AnyCriteria influences AnyCommitment. Determine what influences AnyCommitment. How does AnyCriteria help in judging the deliverable? On what basis? The government dictates state law, and everyone in the state is expected to follow it.

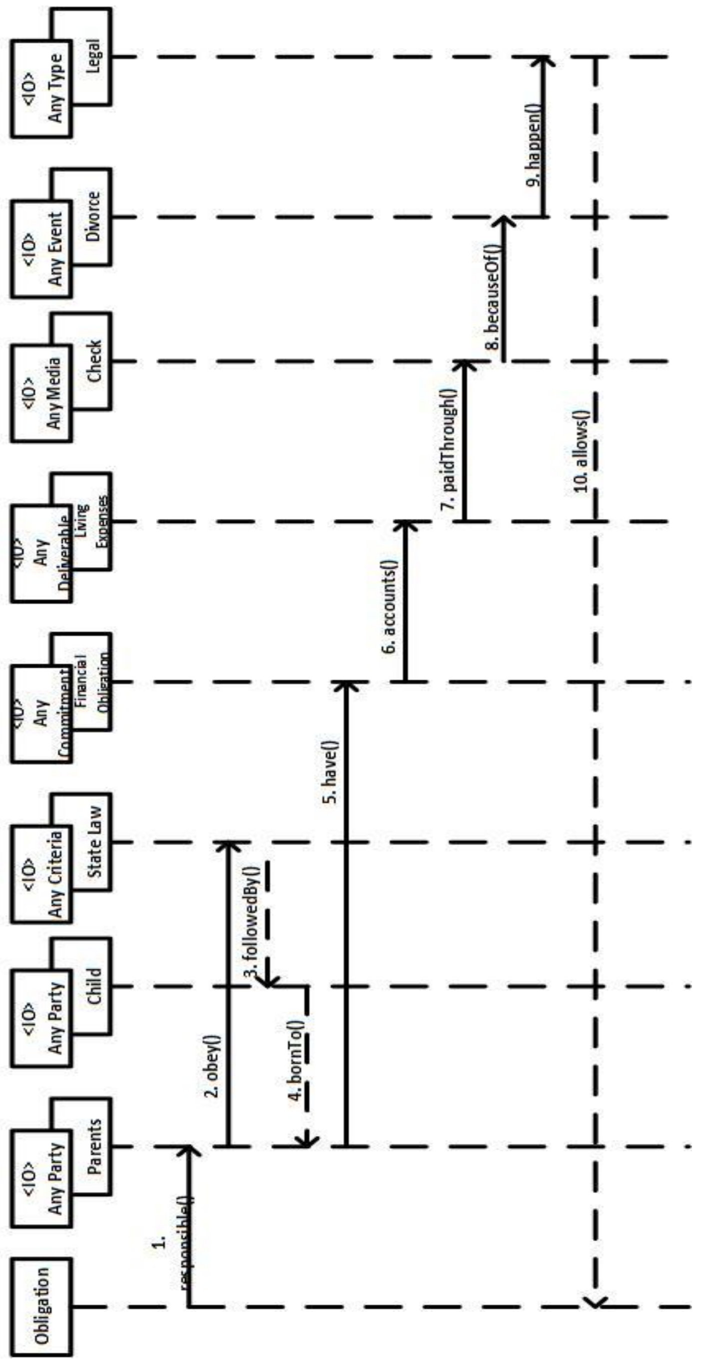


Figure 25. Parent-Child Sequence Diagram

Summary.

The midsize template provided in this paper exhibits how the core knowledge of AnyCommitment can be used for numerous applications. With proper identification and detection, it is possible to cull out the main theme for the pattern along with its associated BOs and IOs. It also interprets an SDP for AnyCommitment in an effective manner. This model can be extended to identical contexts. The stability, reusability, and robustness of the pattern itself is a very big contribution apart from the reduced time, money, and effort to create the required patterns.

AnyComplaint SDP

Pattern name. As discussed earlier, a complaint has a certain duration; therefore, it has a beginning and an end. According to the SSM, these characteristics classify complaints as a BO that is used to signify displeasure, anguish, and grievance.

Other names for the term *complaint* are objection, grievance, protest, criticism, accusation, grumble, whine, lament, and gripe ("Complaint," 2016). The main reason for choosing the term *complaint* for this pattern is its applicability to almost all aspects of displeasure. Displeasure, which is the EBT for AnyComplaint, helps in categorizing the other BOs that assist us in realizing the core knowledge.

Context. Complaints can be given in different forms. Some of the most common methods are to complain in person or in writing. Complaints reveal the level of displeasure that one encounters during diverse situations from being poorly treated at a store or from general unhappiness with the government. The following are some of the scenarios where we can apply the AnyComplaint SDP.

Sometimes a customer (AnyParty) goes to a store such as Walmart (AnyParty) to buy a product like a toy (AnyEntity), only to discover that the toy is defective (AnyReason). The buyer returns to Walmart (AnyParty) to file a complaint (AnyComplaint) in person about the product (AnyEntity). The Customer (AnyParty) informs the store about the defect, such as a broken wheel on a toy car and so it is a performance-related complaint (AnyType). According to the Wal-Mart store policies (AnyRule), their representatives resolve the issue by giving the customer a new toy or by returning the customer's money. If the store employees do not resolve the issue, the consumer can write a complaint letter (AnyMedia) or write an email (AnyMedia) to the management and other authorities stating their grievance and providing supporting evidence. These complaints are stored in company records (AnyLog) for reference purposes.

In another scenario, a client (AnyParty) faces connectivity issues (AnyReason) with the Internet because of a bad modem, leased from a provider such as Comcast (AnyParty). The client contacts the customer support team via phone call (AnyMedia) and complains (AnyComplaint) to Comcast about the faulty equipment. He or she could demand a new modem (AnyEntity) because the current one is not working as expected. In this case, there is a service defect (AnyType) related to Comcast's equipment. The customer care representative first asks for user details such as an account number and service address and then reviews the contract (AnyRule). The representative places a note (AnyLog) concerning the problem in the client's records and asks the client to exchange the modem for one that works properly.

Problem. The existing problem is that there are no SDPs existing for AnyComplaint that define the core knowledge of a complaint. Once we have an SDP for AnyComplaint with displeasure as the EBT, then we can build as many applications as we want by using it repeatedly. The pattern needs to be reusable without changing the main core. Consider the following functional and non-functional requirements.

Functional requirements.

- The ultimate cause of AnyComplaint is displeasure. Displeasure is the EBT of the pattern AnyComplaint.
- Complaints can be filed against AnyParty. AnyParty has a name, contact information, and at least two roles. It can be a country, an institution, or an organization.
- Complaints are also filed against AnyActor. The consumer, who files a complaint, is also AnyActor. AnyActor has a name, a unique ID, a role, and a category. It has operations such as playRole(), interact(), request(), explore(), and receive().
- AnyComplaint is filed because of unacceptable and objectionable conditions. There could be several reasons for AnyComplaint. AnyComplaint has a reason and is governed by a set of rules. It has a date, supporting evidence, and AnyReason for the complaint. When a person complains about a product, he or she would demand() compensation for the problem with it. AnyComplaint has a status() such as solved or not solved. It also has a description() mentioning the details of the complaint (Klein, 2008).

- AnyRuleis formed because of different factors. AnyParty or AnyActor follows a certain set of rules. AnyComplaint is filed when one violates those set rules. AnyRule has a name, a unique ID, and a description. AnyRule is made for others to follow(), and it should be allowed() by higher authorities such as the government. AnyRule must be accepted() by higher authorities to be followed and controlled().
- AnyReason is influenced by some conditions and requirements. AnyComplaint has AnyReason, and different types of AnyReason exist. AnyReason has attributes such as description(), describes(), and decides(). AnyReason often has a description(). AnyReason helps one decide() and give conclusions.
- AnyType reason exists based on the complaint and entity or event. AnyType has a name, an ID, a property, and possibly a subtype. It has operations such as change(), operateOn(), subtype(), and classify(). AnyEntity have AnyType.
- AnyEntity has a complaint filed against it. AnyEntity could be a product or service. When AnyActor or AnyParty is not satisfied with AnyEntity, then it complains. AnyEntity has a name, an id, and a type. AnyEntity is expected to performFunction(). Other operators are relationship(), status(),new(), and old().
- AnyComplaint can be filed against AnyEvent, such as a music concert, or a marriage. When AnyParty is dissatisfied with AnyEvent, there is the possibility of AnyComplaint being filed. AnyEvent has a name, type, and result. AnyEvent occurs() during a particular timeFrame() and is performedBy() AnyEntity to produce a result().

- AnyMedia is the medium through which complaints are filed. A complaint is usually filed through AnyMedia such as a phone call, a complaint letter, or an email. AnyMedia has a name, id, and category. A medium connects() AnyEntity, AnyParty, and AnyEvent to each other. It allows AnyComplaint to be logged by giving access() to the complaint. AnyMedia can help to broadcast()complaints and display() them.
- AnyLog stores complaints given on AnyMedia. AnyLog is specific to the media and may have subdivisions. AnyLog allows users to search for a specific complaint. It has attributes such as id, name, numberOfEntries, and references. AnyLog uses operators such as stored(), search()insert(), and extend().

Non-functional requirements.

- A complaint should be made on time. The sooner one complaints, the higher the chances of it being resolved quickly. Hence, it is essential for the pattern to be timely. In most cases, for example, there are time limits for filing lawsuits. Typically, one can file a lawsuit within two years from the date of the occurrence. In some cases, an individual may have only sixty days to file a lawsuit.
- A complaint should be justifiable. One cannot simply complain without having a proper reason or evidence supporting their complaint. A complaint needs to be reasonable. Consider a scenario where one needs to return a shirt to a store. According to that store’s policy, a consumer has forty-five days to return the purchased item. If the consumer decides to return the shirt after forty-five days,

his return is not justifiable according to store policy. One should check whether the complaint is justifiable after verifying any terms and condition that apply.

- A complaint needs to be well-defined when brought to the attention of a company. The details of the complaint should be complete so that the problem is clearly understood. A complaint should be backed by verifiable facts. Elements that led to the complaint and the desired outcome of the complaint should be clearly stated. If the complaint is well-defined, all the information necessary for judging it will be available.
- If a complaint is not relevant to a scenario, then it cannot be resolved. If the displeasure is caused for one reason but a complaint is filed for an unrelated reason, then it is not relevant. A complaint should be relevant according to the legal system which the complainer resides under (Miller, Harvey, & Parry, 1998). If a person gets out of bed and stubs his toe, then goes to the coffee shop and complains to the manager about it, he is not relevant. Although he could say that not having his coffee earlier caused the accident, the fault lies entirely on himself.

Solution. This these provides a solution to resolving complaints by utilizing the SSMto extract the core knowledge of AnyComplaint. It focuses on how a complaint needs to be given, namely, at the time a person is displeased by AnyActor or AnyParty. The given solution also provides an SDP that allows others to build numerous applications on cross-platform domains. Figure 26 shows the class diagram of the AnyComplaint SDP.

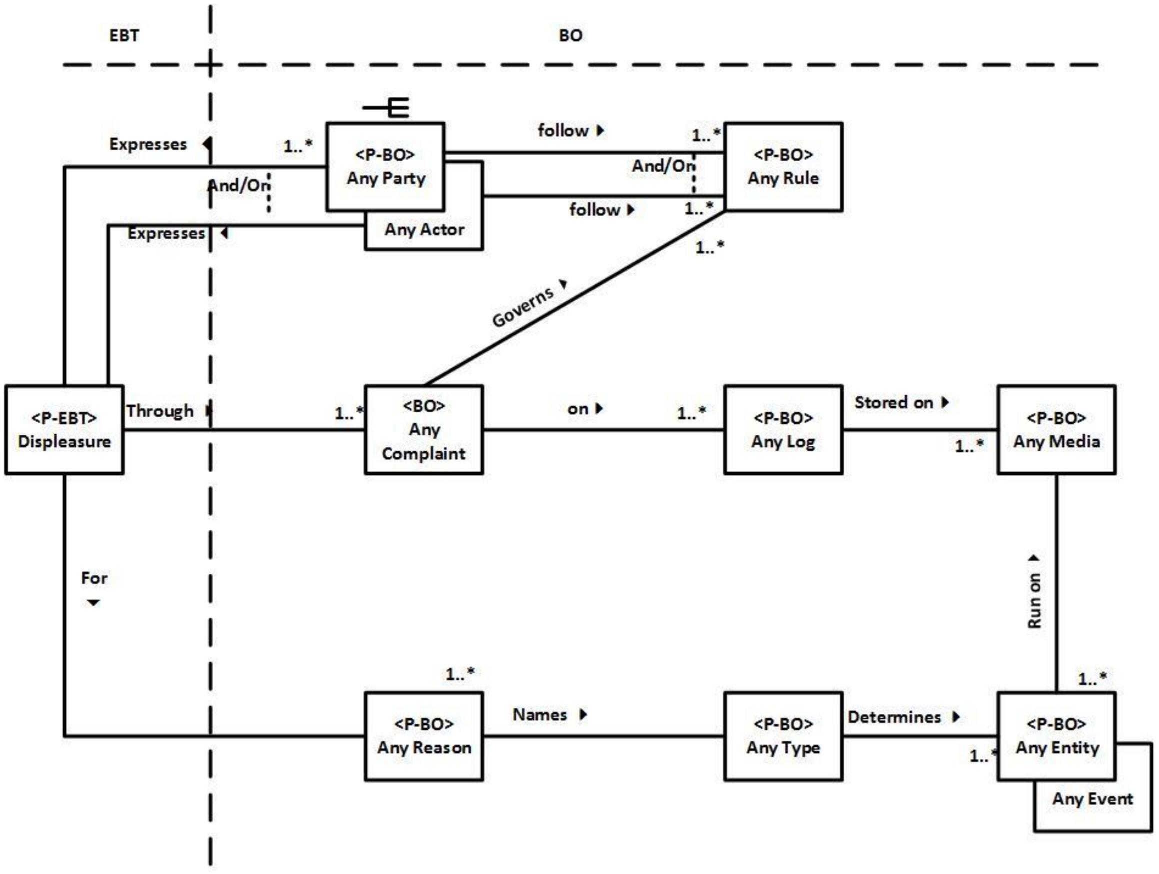


Figure 26. AnyComplaint SDP

Class diagram description.

- AnyActor expresses Displeasure.
- AnyParty or AnyActor follows AnyRule.
- AnyParty or AnyActor files AnyComplaint because of Displeasure with AnyEntity.
- AnyComplaint is governed by AnyRule.
- AnyComplaint has AnyReason named by AnyType.

- AnyComplaint is given about AnyEntity.
- AnyType determines AnyEntity.
- AnyMedia is used to give AnyComplaint.
- AnyLog is used to store AnyComplaint.

Applicability.

Table 23
Applicability of the Complaint EBT Across Several Disciplines

BO	Cafeteria	Late Delivery	Slow Internet	Landlord	Flight Tickets
AnyParty	Barista, customer	Dell, patron	Comcast, Internet User	Landlord, tenant	Webjet, client
AnyComplaint	Bad crink	Compensat e	Fail to respond	Maintain	Travel delay
AnyReason	Incorrect mix	Lost	Poor Internet connection	Dirty house	Misleading terms and conditions
AnyEntity	Coffee	Laptop	Modem	Carpet	Air Fare
AnyType	Wrong elements	Delivery	Internet connection	Property	Flight tickets
AnyRule	Labels	Shipping policy	Perform-ance test	Lease	Terms and conditions
AnyMedia	In person	Phone call, e-mails	Phone calls	Letter	Phone call
AnyLog	Company records	Order tracking logs	Network packet records`	Phone records, tenant records	Airline complaint records

2.11: AnyViolation SDP

Case Study: Complaint at a Store

A customer (AnyParty) goes to a shoe store (AnyParty) and buys a pair of running shoes (AnyEntity). However, the footwear (AnyType) purchased gives the consumer blisters (AnyReason) and scratches (AnyReason). So, the shopper goes back to the store in person (AnyMedia) and demands an exchange (AnyComplaint). It is a store policy (AnyRule) to exchange bad products. All the details about the shopper, shoes, and exchange are stored in the store's records (AnyLog). Figure 27 provides a class diagram of this case study.

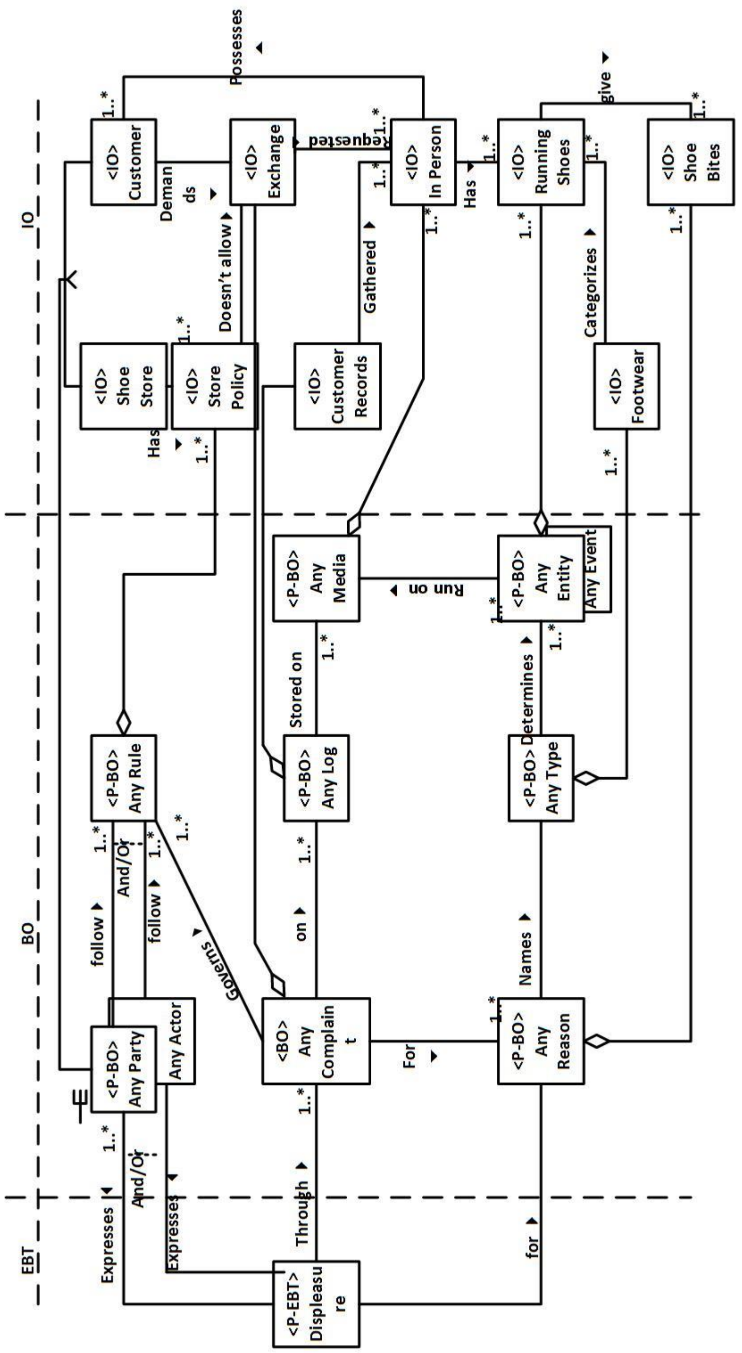


Figure 27. The class diagram for a shoe store customer using EBts, BOs, and IOs.

Use case and description. Consider the use case for filing a complaint with a shoestore in Table 24.

- Use Case ID: 002
- Use Case Name: Filing a Complaint With a Shoestore

Table 24
Use Case 2 for Complaint

Actor	Role
AnyParty	Store, customer

Class	Type	Attributes	Operations
Displeasure	EBT	domain, name, context, id, place	annoy()
AnyParty()	BO	name, phoneNumber, email, type, address	buyShoe()
AnyRule()	BO	number, name, type , duration, regulation	provideGuideline()
AnyComplaint()	BO	number, description, status, product, statement	accuse()
AnyReason()	BO	description, name, proof, limit, case	provide()
AnyEntity	BO	name, code, type, position, state, status, isAlive	determine()

AnyEvent()	BO	name, id, type, occasion, contract, contender	compete()
			appear()

AnyType()	BO	property, subtype, interfaceList, methodList, name	classify()
AnyMedia()	BO	securityLevel, name, sector, mode, tool,	broadcast()

		method	

AnyLog()	BO	limitation, path, reference, size, numberOfEntries, date	find()
			search()

AnyParty(Shoe Store)	IO	name, phoneNumber, email, type, address	has()
AnyParty(Shopper)	IO	name, phoneNumber, email, type, address	demands(), posessedBy()
AnyRule(Store Policy)	IO	number. name, type, duration, regulation	doesNotAllow()
AnyComplaint(Exchange)	IO	number, description, status, product, statement	requestedBy()
AnyReason(Shoe Bite)	IO	description, name, proof, limit, case	receivedFrom()
AnyEntity(Running Shoes)	IO	name, code, type, position, state, status, isAlive	sortBy(), give()
AnyType(Footwear)	IO	id, property, subtype, interfaceList, methodList, name	categorizes()
AnyMedia(In Person)	IO	securityLevel, name, sector, mode, tool, method	displayed()
AnyLog(Customer Records)	IO	limitation, path, reference, size, numberOfEntries, date	gathers()

Use case description.

- Sore feet (AnyReason) causes the customer (AnyParty) Displeasure. (TC: Who expresses the displeasure? Because of what does the displeasure exists? Who is annoyed by the displeasure? What lead to the annoyance? How is displeasure put forward?) The customer files a complaint with the store manager.

- The customer (AnyParty) buys shoes and feels unsatisfied with the purchase if the shoe does not fit well. The customer (AnyParty) also follows store policy (AnyRule) when he or she expresses() displeasure. (TC: What is the reason for the buyer's dissatisfaction? Which pair of shoes does AnyParty buy? Why does AnyParty feel unsatisfied?) Every customer must follow the store policy when making returns.
- The customer (AnyParty) must follow store policy (AnyRule). The store can provideGuideline() to the customer (AnyParty) (TC: On what basis were the guidelines created? Who monitors these guidelines? Who follows AnyRule? What guideline does AnyRule provide?) Store Policy does not allow exchanges and so a buyer cannot exchange faulty shoes.
- The customer (AnyParty) wants to exchange (AnyComplaint) the shoes because the shoes do not fit correctly (AnyReason). The customer accuses the store (AnyEntity) of poor business practices. (TC: What reason is given for AnyComplaint? Who accuses AnyEntity? What is the accusation about?) The customer wants to exchange the shoes.
- Shoe discomfort is the reason (AnyReason) the customer wants to exchange the shoes (AnyComplaint). (TC: Why is AnyReason provided? What is the reason for AnyComplaint? Who provides a reason?) Blisters and cuts occur when wearing poorly fitting shoes.

- The shoe store (AnyEntity) sells footwear (AnyType). (TC: What determines the type of store? Who determines the type of complaint? Is there any other way AnyEntity can be defined by AnyType?) The store sells high-quality shoes.
- The customer (AnyParty) broadcasts complaints (AnyComplaint) about the shoes in person (AnyMedia). (TC: What does AnyMedia broadcast? Can there be more than one medium?) The store manager listens to the customer complaints.
- A database of customer records (AnyLog) is used to store and find AnyComplaint() (TC: What is AnyLog used to store and find? What else can be stored in AnyLog? How are records searched in AnyLog?) Customer records are used to store customer information.

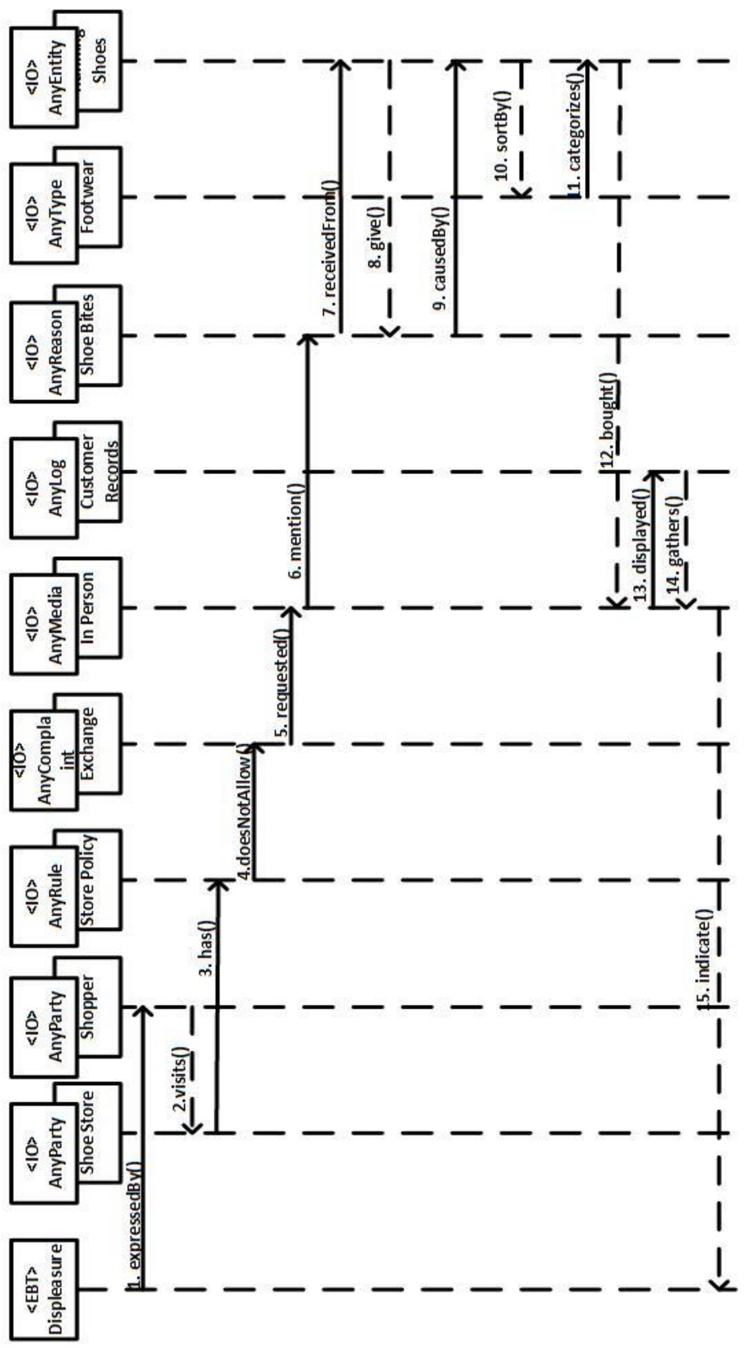


Figure 28. The sequence diagram for AnyComplaint.

Conclusion. The mid-size template provided in this paper exhibits how the core knowledge of AnyComplaint can be used for numerous applications. It interprets an SDP for AnyComplaint in an effective manner. This model can also be extended to identical contexts. The pattern's reusability is an important contribution to design.

AnyDeed SDP

Pattern name. AnyDeed is a BO. It is something that is done or performed. AnyDeed is something that is accomplished, finished, compassed, executed, or undertaken. Other words for deed are act, adventure, feat, action, and stunt ("Deed," 2016).   This term is general in that it is appropriate for all the possible scenarios of AnyDeed. This leads to an SDP for AnyDeed.

Context. AnyDeed has applications in different domains such as business, education, medicine, and sports. Some types of deed are consent AnyDeed, declaratory AnyDeed, default AnyDeed, divine AnyDeed, and investigative AnyDeed. The following are some of the contexts where we can apply AnyDeed as an SDP.

In real estate, a title deed (AnyDeed) serves as proof (AnyEvidence) to the buyer (AnyParty) that the seller (AnyParty) owns the right to sell (AnyLaw) the property. The deed acts as a guarantee (AnyReason) that the property (AnyEntity) does not have any debts associated with it. In addition, the seller must sign a disclosure stating any known problems with the property. The history of the property is not covered in the title deed. Problems that previous owners had with the property are not listed in closing documents and title deeds. To help prevent large expenses after the purchase of the property, a special (AnyType) warranty is needed.When closing documents are complete, the buyer, seller, and title company keep records (AnyLog) of all the closing papers (AnyMedia) associated with the property.

Or consider that a criminal deed (AnyDeed) is something that is done against the law. Theft (AnyEvent) is a criminal deed and can be categorized on different levels. One level of theft is pickpocketing (AnyType). The level of theft depends on the value of property stolen and whether or not weapons were involved. Consider a situation where a criminal (AnyParty) steals the wallet of a pedestrian (AnyParty), thereby violating (AnyReason) state law (AnyLaw). If the police catch the criminal, then the offender is punished by the court in person (AnyMedia). The victim reveals information (AnyEvidence) about the crime. The court recorder makes a copy of all the proceedings (AnyLog) so that others can use it for reference.

Problem. As of now, there is not an SDP existing for AnyDeed that defines the core knowledge of it. Once we have an SDP for AnyDeed, then we can build many applications as many applicationsas we want by using it repeatedly. The following are functional and non-functional requirements of the SDP.

Functional requirements.

- Recording represents anything that records. It has attributes such as context, isRecording, and type. It has operations such as copy(), document(), and register().
- AnyDeed represents doing something. It has attributes such as actName, type, and context. This class has operations such as execute(), accomplish(), and perform().

- AnyParty represents the party that is involved in the deed. A party can be a country, political party, organization, or a person affiliated with an organization. AnyParty has attributes such as name, id, and location. It has operations such as participate(), collectData(), and interact().
- AnyLaw represents the laws that let AnyParty or AnyActor know what is allowed and what is not. Some attributes are standard, method, and lawName. It has operations such as prevent(), control(), and regulate().
- AnyEvidence supports the deeds. It uses attributes such as description, name, and id. The operations are prevent(), test(), and measure().
- AnyType represents the type of deed. AnyType has attributes such as id, name, interfaceList, methodList, and property. A few of the operations are change(), categorize(), and subtype().
- AnyMedia represents the media though which AnyDeed occurs. Data and information reside inside this medium. It has attributes such as name, availability. and isAvailable. It has operations such as display(), store(), capture(), broadcast(), and connect().
- AnyDeed is done for AnyEntity and is named by AnyType of AnyDeed. AnyEntity has attributes such as id, name, and status. Its operations can be update(), relationship(), and type().
- AnyDeed is done for AnyEvent and is named by AnyType of AnyDeed. AnyEvent's attributes are name, occasion, type, and outcome. Its operations can be to appear(), happen(), and arrange().

- AnyLog represents the recording of AnyDeed. It could be a file, digital file, or any kind of storage. AnyLog allows users to search for a specific deed and has attributes such as id, name, numberOfEntries, and references. It has operations such as search(), insert(), and extend().

Non-functional requirements.

- AnyDeed should be complete in all respects because an incomplete deed is of no use. Consider a situation in which a professor is teaching his student a lesson, and the professor teaches only half of the lesson. In that situation, the students would not understand the lesson completely. AnyDeed must be complete.
- AnyDeed should be achievable. For example, while developing a product, one should not place impossible goals on themselves. If it is a software code, a developer gets a lot of errors and warnings. The developer must remain persistent and debug or rebuild the software. AnyDeed should be real regarding achievability. A clear and well-explained deed is easy to execute.

Solution. This paper provides a solution by utilizing the SSM to extract the core knowledge of AnyDeed. It also focuses on how AnyDeed needs to be executed. The solution provides an SDP that incorporates various methods for applying AnyDeed. Figure 29 shows the class diagram for the AnyDeed SDP.

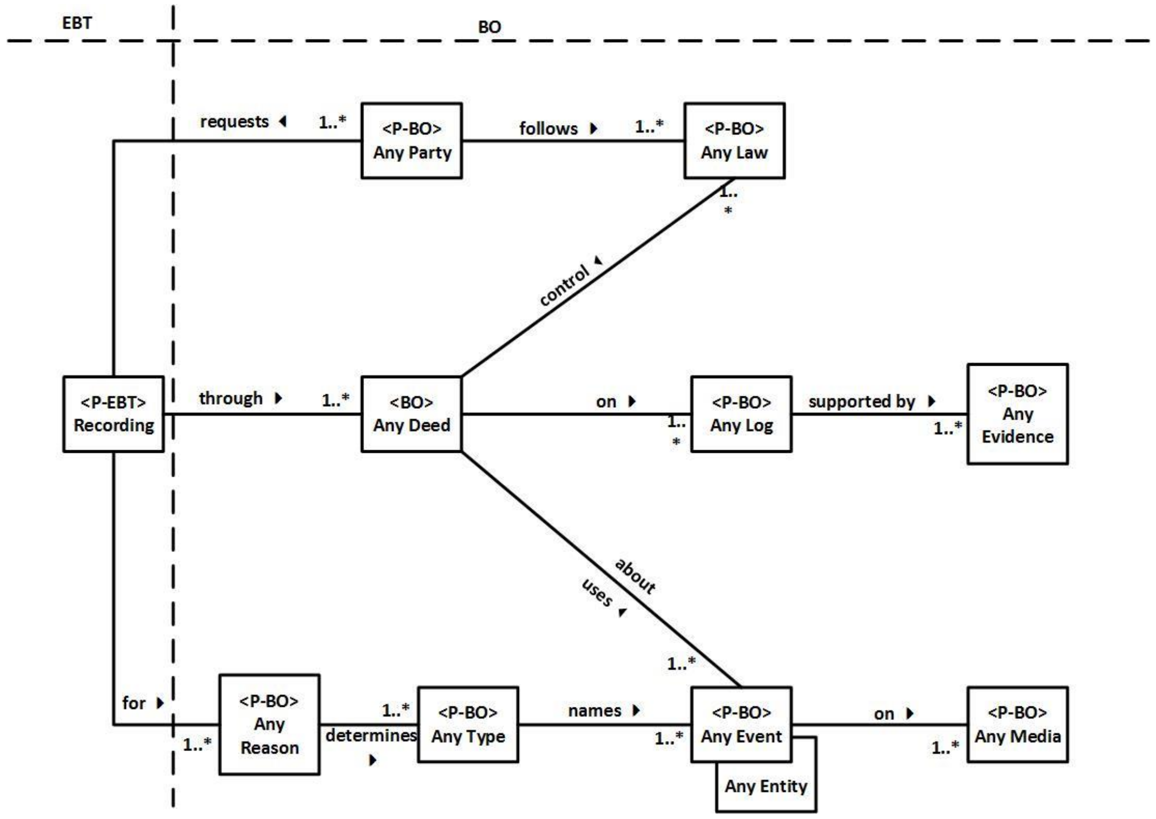


Figure 29. The AnyDeed SDP

Class diagram description.

- AnyParty requests AnyDeed for recording a transaction.
- AnyParty follows AnyLaw.
- AnyLaw controls AnyDeed.
- AnyReason determines AnyType.
- AnyType names AnyEntity or AnyEvent.
- AnyEntity or AnyEvent resides on AnyMedia.
- AnyLog is supported by one or more AnyEvidence.

Applicability. The applicability SAP is useful in several contexts. Table 25 explores how the Recording SAP could benefit several situations and how the classes compare in each situation.

Table 25
Applicability of the Recording EBT Across Several Disciplines

BO	Good Deed	Criminal Deed	Marriage Deed	Exam Deed	Rental Deed
AnyParty	Custome r, staff	Criminal, pedestrian	Bride, groom	Student, professor	Landlord, tenant
AnyLaw	Store policys	State law	Marriage contract	College policy	Rental law
AnyDeed	Returns wallet	Criminal act	Religious act	Cheating	Home repair
Any:Log	Storage	File repository	State records	Student records	Lease records
AnyEntity	Wallet				Home
AnyEvent		Theft	Marriage	Exam	
AnyType	Irrespon sible	Criminal	Monogam y	Unauthori zed	Legal
AnyMedia	In-person	Court	Website	Paper	Court
AnyReason	Drops-off	Violation	Compatabl e	Pass the exam	Dispute
AnyEvidence	ID card, credit cards	Informatio n	Certificate	Video	Lease contract

Notes: ID = identification.

Case Study: Good Deed

Consider a situation where a customer (AnyParty) visits a store and loses money. Later, the customer realizes that he or she dropped (AnyReason) his or her wallet (AnyEntity) in the store in an irresponsible manner (AnyType). One of the staff members (AnyParty) finds the wallet and stores it in the safe (AnyLog) because the store policy (AnyLaw) stipulates that any lost and found item should be stored there. When the customer returns to the store in person (AnyMedia) to check for his wallet, the staff verifies his ID (AnyEvidence) and returns the wallet (AnyDeed). Figure 30 provides the class diagram of this case study.

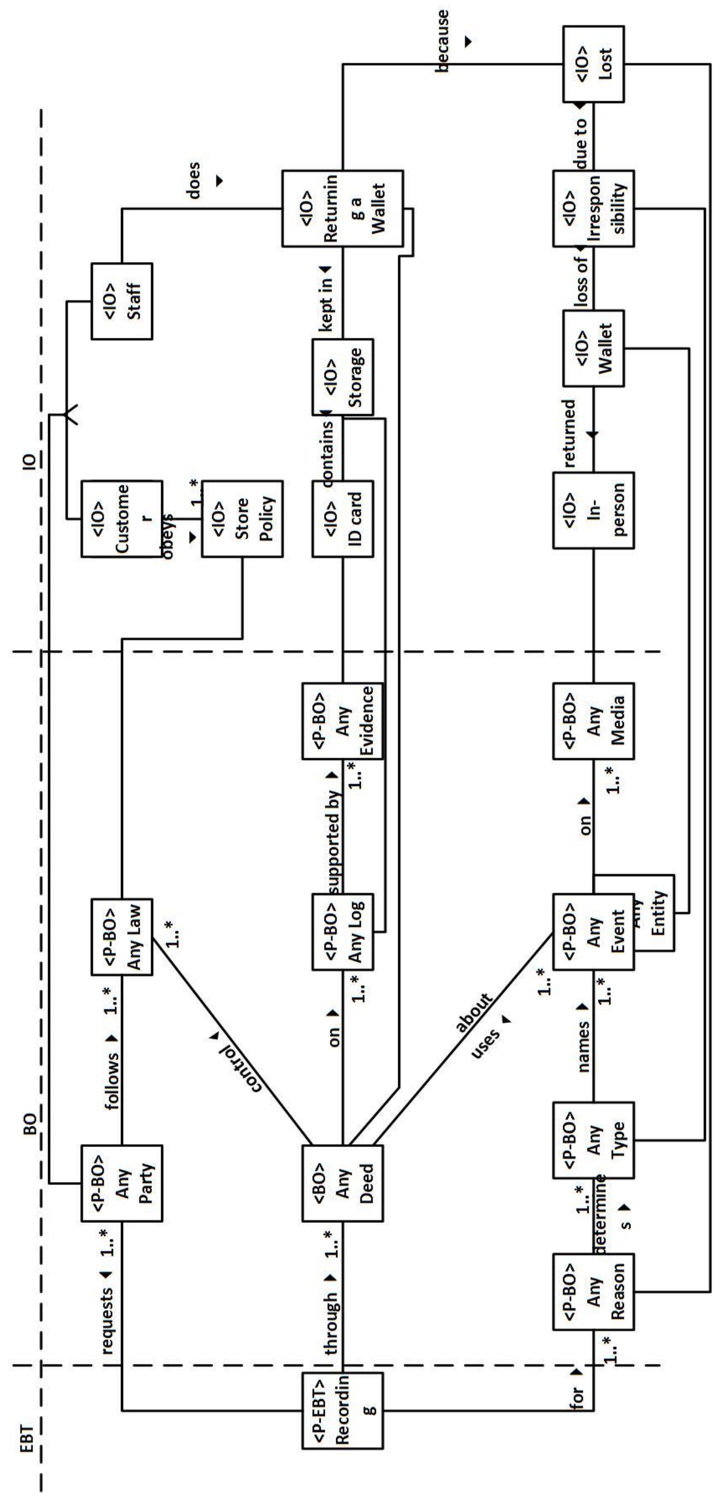


Figure 30. The class diagram for returning a lost item.

Use case and description. Consider the use case for returning a lost item in Table 26.

- Use Case ID: 001
- Use Case Name: Returning a Lost Item

Table 26
Use Case 1 for AnyDeed: Returning a Lost Item

Actor	Role
AnyParty	Customer, staff

Class	Type	Attributes	Operations
Recording()	EBT	domain, name, context, id, state	stores()
AnyDeed()	BO	context, type, level, status, actionTaken	perform()
AnyParty()	BO	name, id, type, phoneNumber, address	participate()
AnyLaw()	BO	id, description, status, product, statement	regulate()
AnyEvidence()	BO	id, description, name, proof, limit, case	prove()
AnyEntity()	BO	name, id, type, position, state, status, isAlive	names()
AnyEvent()	BO	name, id, type, occasion, contract, contender	happen()
AnyType()	BO	id, property, subtype, interfaceList, methodList, name	classify()
AnyMedia()	BO	name, id, category, description, usedFor	resideOn()
AnyReason()	BO	description, proof, justification, id, context	conclude()
AnyLog()	BO	id, name, location,	search()

		numberOfEntries, size	
AnyParty(Customer)	IO	dateOfMarriage, name, address, phoneNumber, email	obey()
AnyParty(Staff)	IO	dateOfBirth, name, address, height, hairColor	perform()
AnyDeed(Return a Wallet)	IO	id, status, application, descripton, context	paidThrough()
AnyLaw(Store Policy)	IO	restrictionName, type, severity, threshold, description	followedBy()
AnyEvidence(ID Card)	IO	objective, policy, strategy, id, position	serve()
AnyType(Irresponsibility)	IO	id, property, subtype, interfaceList, methodList, name	lose()
AnyEntity(Wallet)	IO	name, type, capability, status, sector	return()
AnyMedia(In-person)	IO	name, id, category, description, usedFor	meet()
AnyReason(Lost)	IO	description, proof, justification, id, context	resolve()
AnyLog(Storage)	IO	id, name, location, numberOfEntries, size	contain()

The use case is described in detail below.

- Recording() stores AnyDeed(). (TC: What stores AnyDeed()? What types of deeds exist? What is accomplished through AnyDeed()?) A customer requests that the BMV records a vehicle title.
- AnyParty() participates in AnyDeed(). (TC: Who participates in AnyDeed()? How many participants exist?) Every customer obeys store policy.

- AnyDeed() performs something. (TC: What performs AnyDeed()? Who performs a deed? Does the staff perform the deed?) If a customer loses his wallet, it is kept in the safe.
- AnyLaw() regulates AnyDeed(). (TC: What regulates AnyDeed()? Who makes these laws? What are the laws meant for?) The staff follows store policy.
- AnyEvidence() proves AnyDeed(). (TC: What proves AnyDeed()? What information is considered as evidence? Who provides evidence?) The ID card is lost because of irresponsibility.
- AnyType() classifies AnyDeed(). (TC: What classifies AnyDeed()? How many AnyTypes of deeds exist?) Irresponsibility leads to the loss of the wallet.
- AnyMedia() resides on something. (TC: What does AnyMedia reside on? Who selects the medium?) The wallet is returned in-person.
- AnyReason() concludes AnyDeed(). (What concludes AnyDeed()? Is there AnyLogic behind AnyReason()?) The loss of the wallet was due to customer’s irresponsibility.
- AnyLog() searches for AnyDeed(). (TC: What searches for AnyDeed()? Can a diary be considered as a log?) The safe contains the wallet with an ID card in it.
- AnyEntity() names AnyDeed(). (What names AnyDeed()? What is the nature of AnyEntity()?) The wallet is returned to the customer in person.

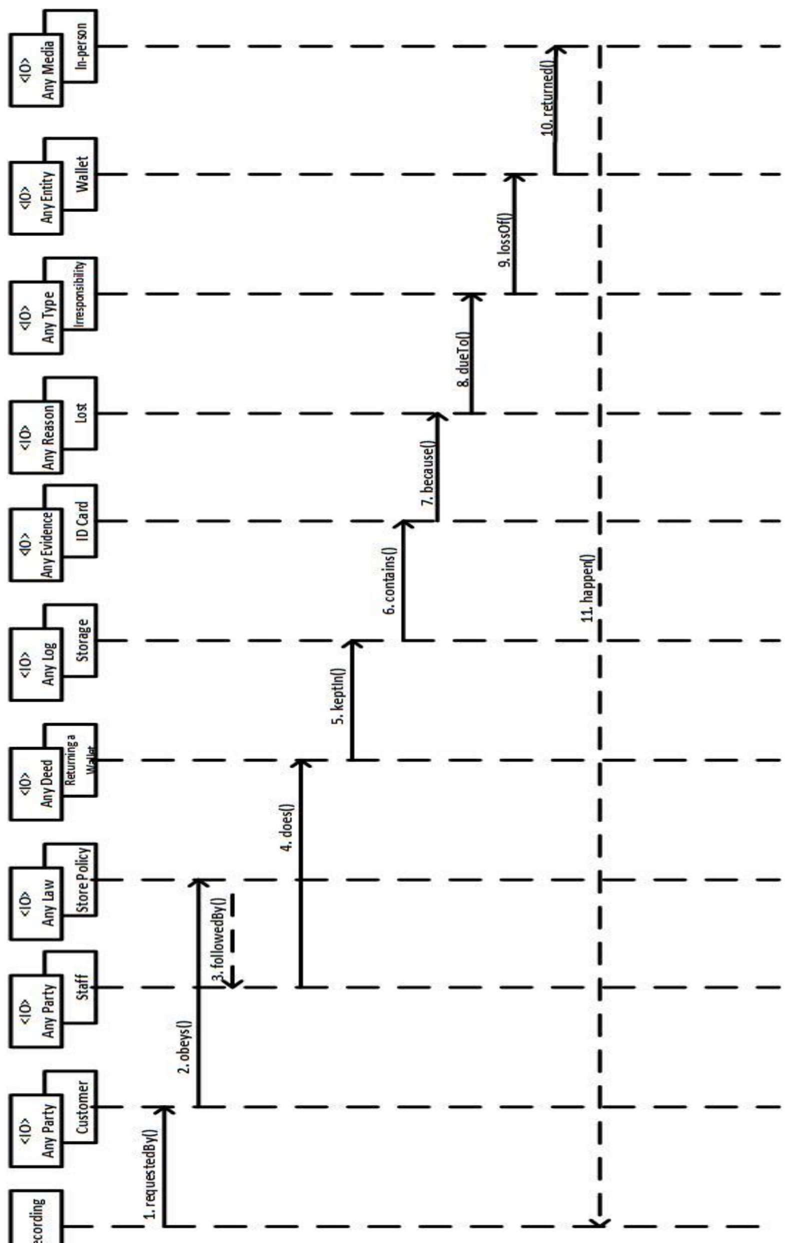


Figure 31. The sequence diagram for returning a lost item.

Summary. The mid-sized template provided in this paper exhibits how core knowledge of AnyDeed can be used for numerous applications. It also interprets SDP for AnyDeed in an effective manner. This model can also be extended to identical contexts. Stability of the pattern itself is a very big contribution apart from savings in time, effort and money. This pattern was created after identifying the EBT for the pattern. AnyDeed is not only robust and flexible, but it is also enduring and extendable to any number of applications.

AnyRate SDP

Pattern name. AnyRate is a BO. It is the price that people pay for a good or service. Rate can be a measure, quantity, frequency, bill, or salary. Other words for rate are amount, estimate, price, charge, and speed ("Rate," 2016). Generality is the main reason for choosing this term because it is appropriate for all the possible scenarios of AnyRate. This leads to an SDP for AnyRate.

Context. AnyRate has applications in different domains such as mathematics, science, finance, and medicine. Some types of rate are consent birth rate, billing rate, mortality rate, and birth rate. The following are some of the contexts where we can apply AnyRate as an SDP

Consider a scenario where a freelancer (AnyParty) obeys a certain code of conduct (AnyRule) that is stipulated by a website (AnyMedia). A freelancer is a person who performs assignments (AnyEntity) for a company without directly being associated with the company. The client (AnyParty) also follows these rules. The Client settles the milestone payments (AnyRate) corresponding to work progress (AnyIndicator). All clients have accounts (AnyLog)that store their personal information. A client may give a web development (AnyType) project to a freelancer to complete within a set time so that he is satisfied once he finds that the work is satisfactory (AnyReason).

In another scenario, the crime rate (AnyRate) determines the number of crimes occurring at a location. Consider a location where there are a lot of victims (AnyParty) of aggravated assaults (AnyReason), an unlawful attack by one person toward another person that causes serious injury or death. The police take suspects (AnyParty) into custody in a manner that is permitted by law, but the criminal activities continue (AnyIndicator); this indicates a high crime rate and a failure of the law (AnyRule). The police keep a record (AnyLog) of all the unlawful (AnyType) activities. The news channels (AnyMedia) make the locals aware of the high crime rate and help in spreading awareness.

Pattern. Now, there is no SDP existing for AnyRate that defines its core knowledge. Once we have an SDP for AnyRate, then we can build many applications as many applicationsas we want by using it repeatedly. Below are the functional and non-functional requirements.

Functional requirements.

- Rating represents the measurement of how good something is. It has attributes such as context, numberOfRatings, and type. It has operations such as judge(), categorize(), and earn().

- AnyRate represents the amount charged for something or the speed at which something happens. AnyRate has attributes such as ratio, fee, and context. This class has operations such as assess(), estimate(), and rank().
- AnyParty represents the party that is involved in rate. A party could be a country, political party, organization, or a person affiliated with an organization. AnyParty has attributes such as name, id, and location. It has operations such as participate(), collectData(), and interact()
- AnyRule represents the laws that let AnyParty or AnyActor know what is allowed and what is not. It has attributes such as standard, method, and ruleName. It has operations such as prevent(), control(), and regulate().
- AnyIndicator determines the rate. It has attributes such as description, name, and id. The operations can be signify() and display().
- AnyType represents the type of rate. AnyType has attributes such as id, name, interfaceList, methodList, and property. The operations are change(), categorize(), and subtype().
- AnyMedia represents the media though which AnyRate takes place. Data and information reside inside this medium. AnyMedia has attributes such as name, availability, and isAvailable. It has operations such as display(), store(), capture(), broadcast(), and connect().
- AnyRate is done for AnyEntity, and AnyType of AnyRate names AnyEntity. AnyEntity has attributes such as id, name, and status. Its operations can be to update(), relationship() and type().

- AnyEntity costs AnyRate. It has attributes such as name, occasion, type, and outcome. Its operations can be to appear(), happen(), and arrange().
- AnyLog represents the recordkeeping aspect. It could be a file, data file, or any other type of storage. AnyLog allows us to search for a specific rate and has attributes such as id, name, numberOfEntries, and references. It has operations such as search(), insert(), and extend().
- AnyRate occurs because of AnyReason. This class represents the reason for selling. It has attributes such as description, proof, and justification. It has operations such as conclude(), examine(), and resolve().

Non-functional requirements.

- AnyRate should be complete. AnyRate should be final. For example, in the freelancing sector, a client should finalize the rate with the freelancer for the service rendered prior to rendering it; this agreement helps both parties in the event of a future problem.
- AnyRate has to be measurable and quantifiable so one can judge the value that he or she is getting. For example, while buying a property for rental in different areas, it is important to look at the existing rental rates. The rate depends on the normal range of rental rates in the area. Usually, the rent also varies according to the type of property such as condos, apartments, and commercials buildings.
- AnyRate should be acceptable. In the example given above, the rental rate has to be reasonable so that it is acceptable by a person who rents the space. Otherwise,

the future occupant looks somewhere else. An exorbitant rate deters anyone from occupying the property. Hence, a rate should be acceptable.

- AnyRate should rise in a progressive manner over time. Inflation is a convention that is followed throughout the world. In the case of real estate, the annual rent increases progressively every year to reflect the cost of living as well as the cost of maintaining the property.
- AnyRate should be true. It should reflect the current standards and follow the existing rules and regulations. It should not be subject to additional fees or extra payments. Since rate denotes something that is measurable and quantifiable, it should be true and should reflect the actual measure.

Solution. This thesis provides a workable and qualified solution by utilizing the SSM to extract the core knowledge of AnyRate. It also focuses on how AnyRate needs to be provided as a standard measure. The given solution provides an SDP that incorporates various methods for applying AnyRate and for building numerous applications on cross-platform domains. Figure 32 shows the class diagram of the AnyRate SDP.

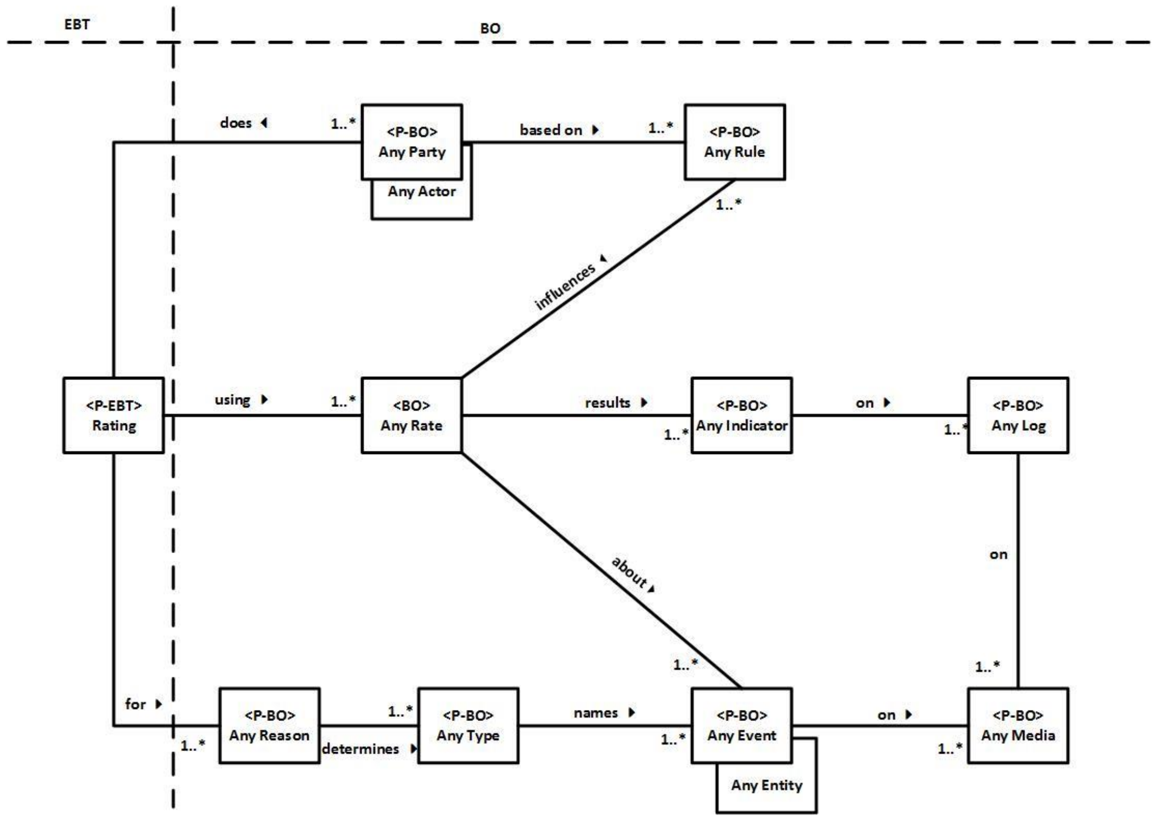


Figure 32. Class diagram of the AnyRate SDP

Class diagram description.

- AnyParty rates using AnyRate.
- AnyParty is based on AnyRule.
- AnyRule influences AnyRate.
- AnyReason determines AnyType.
- AnyType names AnyEntity or AnyEvent.
- AnyEntity or AnyEvent resides on AnyMedia.
- AnyRate results in AnyIndicator.

- AnyIndicator supports AnyLog.

Applicability.

Table 27
Applicability of the Rating EBT Across Several Disciplines

BO	Freelance	Criminal	Movie	Dance Contest	Hotel
AnyParty	Author, client	Victim, suspect	Director, audience	Student, judge	Hotel, tourist
AnyRule	Contract	Legal	Film regulations	Contest rules	Hotel policy
AnyRate	Payment	Fine	General audience	Entry fee	Five star
AnyLog	Account	Police records	Rating system	Contest records	Database
AnyEntity	Project		Movie		
AnyEvent		Crime		Contest	Rating process
AnyType	Web development	Criminal	Film	Ballet	Internet
AnyReason	Satisfaction	Robbery	Theater Refusal	Win the prize	Vacation
AnyMedia	Website	News Channel	Film	Video	Internet
AnyIndicator	Work progress	Repeat offense	Certificate	Video footage	Ranking

Notes: CPR = Cardiopulmonary Resuscitation; DNR = Do not Resuscitate.

Case Study: Freelancing Rate

Consider a scenario where a freelancer (AnyParty) follows a certain code of conduct (AnyRule) provided by a website (AnyMedia), and the freelancer follows these rules without fail. The client (AnyParty) settles a milestone payment (AnyRate) corresponding to the progress of the work. All clients have online accounts (AnyLog) that store their personal information and payment details. A client gives a web development (AnyType) project to a freelancer to complete within a specified time frame. The freelancer should satisfy the customer (AnyReason) not only with the quality work but by delivering the work in a timely manner. In addition to this, the freelancer should also notify the client about the work progress (AnyIndicator) to provide complete satisfaction (AnyReason). Figure 32 provides the class diagram of this case study.

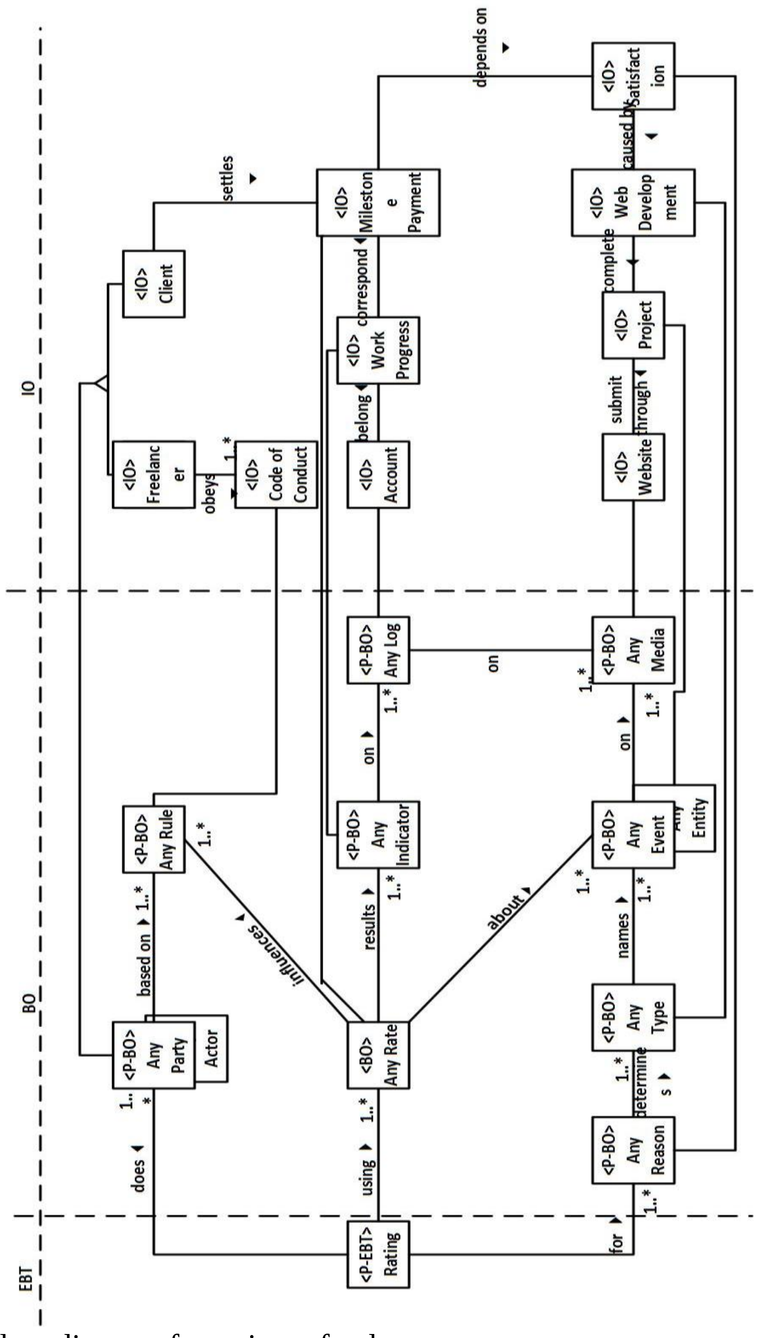


Figure 33. A class diagram for rating a freelancer.

Use case and description. Consider the use case for hiring a freelancer in Table 28.

- Use Case ID: 002
- Use Case Name: Hiring a Freelancer

Table 28
Use Case 2 for AnyDeed: Hiring a Freelancer

Actor	Role
AnyParty	Freelancer, client

Class	Type	Attributes	Operations
Rating()	EBT	numberOfRatings, domain, name, context, id, state	judge()
AnyRate()	BO	ratio, fee, context, type, level, name, status	ranks()
AnyParty()	BO	name, id, type, phoneNumber, address	participate()
AnyRule()	BO	standard, method, name, description, status, product, statement	control()
AnyIndicator()	BO	id, description, name, proof, limit, case	represent()
AnyEntity()	BO	name, id, type, position, state, status, isAlive	signify()
AnyEvent()	BO	name, id, type, occasion, contract, contender, outcome	happen()

AnyType()	BO	id, property, subtype, interfaceList, methodList, typeName	classify()
AnyMedia()	BO	name, id, isAvailable, category, description, usedFor	resideOn()
AnyReason()	BO	description, proof, justification, id, context	conclude()
AnyLog()	BO	id, name, size, location, date, numberOfEntries	search()
AnyParty(Freelancer)	IO	dateOfMarriage, name, address, phoneNumber, email	obey()
AnyParty(Client)	IO	dateOfBirth, name, address, height, hairColor	settle()
AnyRate(Milestone Payment)	IO	id, status, application, description, context	correspond()
AnyRule(Code of Conduct)	IO	restrictionName, type, severity, threshold, description	followedBy()
AnyIndicator(Work Progress)	IO	objective, policy, strategy, id, position	belongsTo()
AnyType(Web Development)	IO	id, property, subtype, interfaceList, methodList, name	complete()
AnyEntity(Project)	IO	name, type, capability, status, sector	submitThrough()
AnyMedia(Website)	IO	name, id, category, description, usedFor, isAvailable	facilitate()

AnyReason(Satisfaction)	IO	description, proof, justification, id, context	causedBy()
AnyLog(Account)	IO	id, name, numberOfEntries, size, location	depend()

- Rating() judges AnyRate(). (TC: What judges AnyRate()? Can Rating be considered as grading?) The rate is given by a freelancer.
- AnyParty() participates in AnyRate(). (TC: Who participates in AnyRate()? Why does a party need the rate? What does a party rate*?)* The client participates in a milestone payment.
- AnyRate() ranks something. (TC: What ranks AnyRate()? Who does the ranking? On what terms is a rank determined*?)* Milestone payments correspond with work progress.
- AnyRule() regulates AnyRate(), (TC: What regulates AnyRate*()?)* A code of conduct is followed by the client.
- AnyIndicator() signifies AnyRate(). (TC: What signifies AnyRate()? How can an indicator symbolize a rate*?)* The work progress is posted on an account.
- AnyType() classifies AnyRate(). (TC: What classifies AnyRate()?) The project is classified as web development.
- AnyMedia() resides on something. (TC: What does AnyMedia reside on? What does a media facilitate?) The website facilitates a rating.
- AnyReason() concludes AnyRate(). (What concludes AnyRate()?) Good web development causes satisfaction.

- AnyLog() searches for AnyRate() (TC: What searches for AnyRate()? Can records be searched using logs?) Satisfied clients give the freelancer positive feedback.
- AnyEntity() names AnyRate() (What names AnyRate()?) The project is submitted through a website.

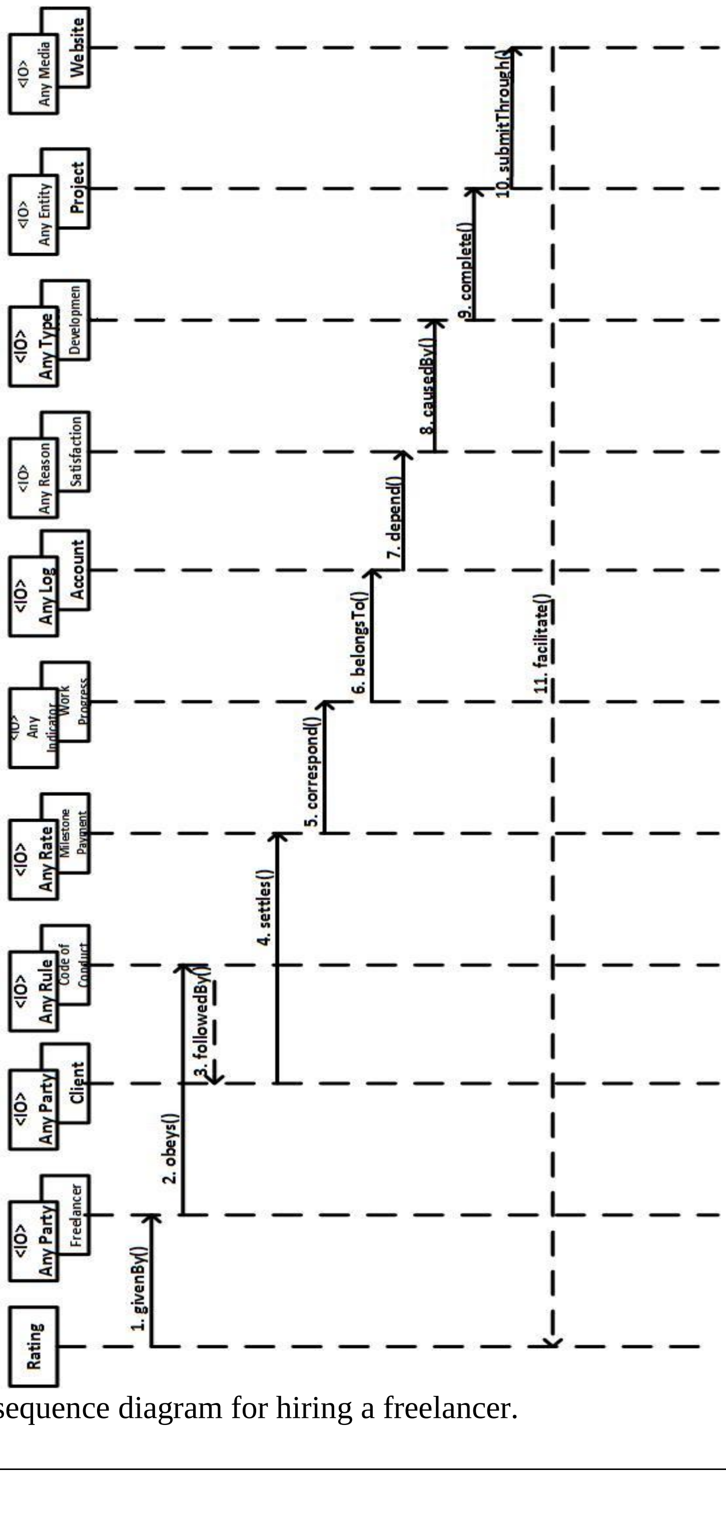


Figure 34. The sequence diagram for hiring a freelancer.

Summary. The midsize template provided in this paper exhibits how the core knowledge of AnyRate can be used for numerous applications. The pattern template provides a brief view of how a developer can extract the EBT for the system and link it with other essential BOs and IOs. The resulting pattern is not only stable and robust, but it can also be used in many other contexts where the AnyRate concept is used. It interprets an SDP for AnyRate in an effective manner. This model can also be extended to identical contexts. The stability of the pattern along with savings in cost, effort, and time are a valuable contribution.

Judgment SAP

Pattern name. Judgment is an EBT. It is a formation of an opinion on something. Judgment is a decision, assumption, conclusion impression, or simply a feeling about something. Other words for judgment are analysis, evaluation, review, verdict, and scrutiny ("Judgment," 2016). Generality is the main reason for choosing this term, as it is appropriate for all the possible scenarios of judgment. This leads to an SAP for judgment.

Context. Judgment has applications in different domains such as law and order, religion, education, and sports. Some types of judgment are consent judgment, declaratory judgment, default judgment, divine judgment, and investigative judgment. The following are some of the contexts in which we can apply judgment as an SAP.

Grades indicate how someone or something performs a task. Consider a situation where a student (AnyParty) writes an exam, and the professor (AnyParty) evaluates (Judgment) the answer sheet (AnyEntity). The professor evaluates the paper by manually reading (AnyMechanism) the answers written by the student. In order to grade (AnyOutcome) the student, the professor uses certain rubrics (AnyLaw) to assess the quality of the student's work. There are several types of grading systems that the professor can choose such as grading based on percentages (AnyType). The professor would note the score given to the student in a roster (AnyForm) and mention comments (AnyOpinion) about the student's performance to indicate where the student made mistakes and how he or she could improve (AnyReason) his or her performance.

Judgment of the court is the decision (Judgment) of the judge (AnyParty) regarding the rights of the parties involved in the case. Consider a situation where the driver (AnyParty) of a car (AnyEntity) is driving his car on a freeway. The driver speeds (AnyReason) and breaks the law. The consequences of a violation of the state law (AnyLaw) are an appearance in court to face the penalty (AnyOutcome). A judgment could be provided in an oral form (AnyForm) in order to conclude (AnyOpinion) quickly. When more than one party is involved in such a case, there is a possibility of arbitration (AnyType), where both parties agree on a settlement (AnyMechanism), and the judge authorizes the decision. The party that wins a case sometimes receives monetary compensation from the other party to recover expenses involved in filing the case.

Problem. Now, there is not an SAP that exists for judgment that defines the core knowledge of a judgment. Once we have an SAP for judgment, then we can build many applications as many applicationsas we want by making the pattern reusable. The following are some of the functional and non-functional requirements.

Functional requirements.

- Judgment represents the opinion one has about something. Judgment has attributes such as context, type, and form. This class has operations such as giveOpinion(), makeDecision(), and conclude().
- AnyParty represents the party that is involved in the violation. A party can be a country, political party, organization, or a person affiliated with an organization. AnyParty has attributes such as name, id, and location. It has operations such as participate(), collectData(), and interact()
- AnyLaw represents the rules formed by a government in a society. It has attributes such as name, createdBy, and penalityForViolation. It has operations such as abide(), adhereTo(), and govern().
- AnyMechanism represents the method used for judgment. It has attributes such as name, context, description, application, and status. It has operations such as execute(), attach(), detach(), and activate().
- AnyReason represents the reason for a judgment. It has attributes such as description, proof, and justification. It has operations such as conclude(), examine(), and resolve().
- AnyType represents the type of judgment. AnyType has attributes such as id, name, interfaceList, methodList, and property. The operations are change(), categorize(), and subtype().

- AnyOutcome represents because the results of the involvement of AnyParty in the judgment. This class has operations such as conclusion, result, and issue. It has operations such as achieve(), complete(), and terminate().
- AnyForm represents one of the many ways in which the judgment can be given. It has attributes such as type, context, and name. This class has operations such as assemble(), design(), and organize().
- AnyOpinion represents the way in which one thinks about something. It has attributes such as description, opinionAbout, and context. It has operations such as assess(), assume(), and feel().
- The judgment is pronounced for AnyEntity. AnyEntity is named by AnyType of judgment. It has attributes such as id, name, and status. Its operations can be to update(), relationship(), and type().
- Judgment is done at AnyEvent. AnyEvent is named by AnyType of judgment. It has attributes such as name, occasion, type, and outcome. Its operations can be to appear(), happen(), and arrange().

Non-functional requirements.

- A judgment should be given on time. Consider a situation where a person is placed in jail for a crime such as a theft but receives punishment ten years after the theft was committed. This means that justice was delayed. The criminal was not punished for years while the wronged party waited for a ruling to determine whether or not they would receive compensation for their loss. Some fast track courts have come into existence for speedy trials for sensitive issues. In another instance, if a pedestrian is crossing a road and suddenly a vehicle approaches the pedestrian at a very fast speed, the pedestrian needs to make the decision to either move backward or move forward. Otherwise, the pedestrian may get injured or die. A judgment must be timely.

- A judgment should be satisfactory and agreeable. When a student is graded based on his performance on a test, the grade received should be such that the student accepts his or her grade. If the student feels that in spite of performing well in the test, he or she has received a low grade, then he or she may not accept the grade and there might be a request for a review of the exam papers. Similarly, if court sentences a person (criminal) without hearing or checking all the evidence, then the criminal may not accept the judgment and appeal against the judgment in higher court. Hence, a judgment should be acceptable by the parties involved.
- Well-defined: It does not make sense when a judgment is not well-defined. A good judgment is expected to be well-defined, so that there is no confusion in regards to the judgment. A well-defined judgment is possible when the reasons behind a judgment are valid. When a person tries to rent an apartment, the person checks if the apartment matches his or her expectations of an ideal house and then judges the apartment based on its qualities. When the person judges, there are reasons why he or she may or may not like the apartment. Those reasons should be well-defined for the judgment to be good. Similarly, when a student receives a grade for his or her performance in class and the professor does not provide proper feedback or comments about the grade, the student may not be satisfied by

the grade because it is not well-defined according to the student. Therefore, even though a judgment is correct, it should be well-defined; otherwise, people might not take it seriously or may refuse to accept the judgment.

- A judgment given should be reasonable and not too harsh or excessive. For example, a court of law cannot sentence a petty thief for a decade of jail time. Similarly, a person who has committed a serious felony should not be given a lighter sentence. In the domain of academic study, a student should be graded based on actual performance. The reason for the judgment should be defined according to the severity of the offense and not on the basis of intuition. Therefore, the reason should be explained clearly.

Solution. This paper provides a workable solution by utilizing the principles of the SSM to extract the core knowledge of judgment. It also focuses on how a judgment needs to be given. The given solution provides an SAP that incorporates various methods to apply judgment and to build numerous applications on cross-platform domains. Figure 34 shows the class diagram of Judgment SAP.

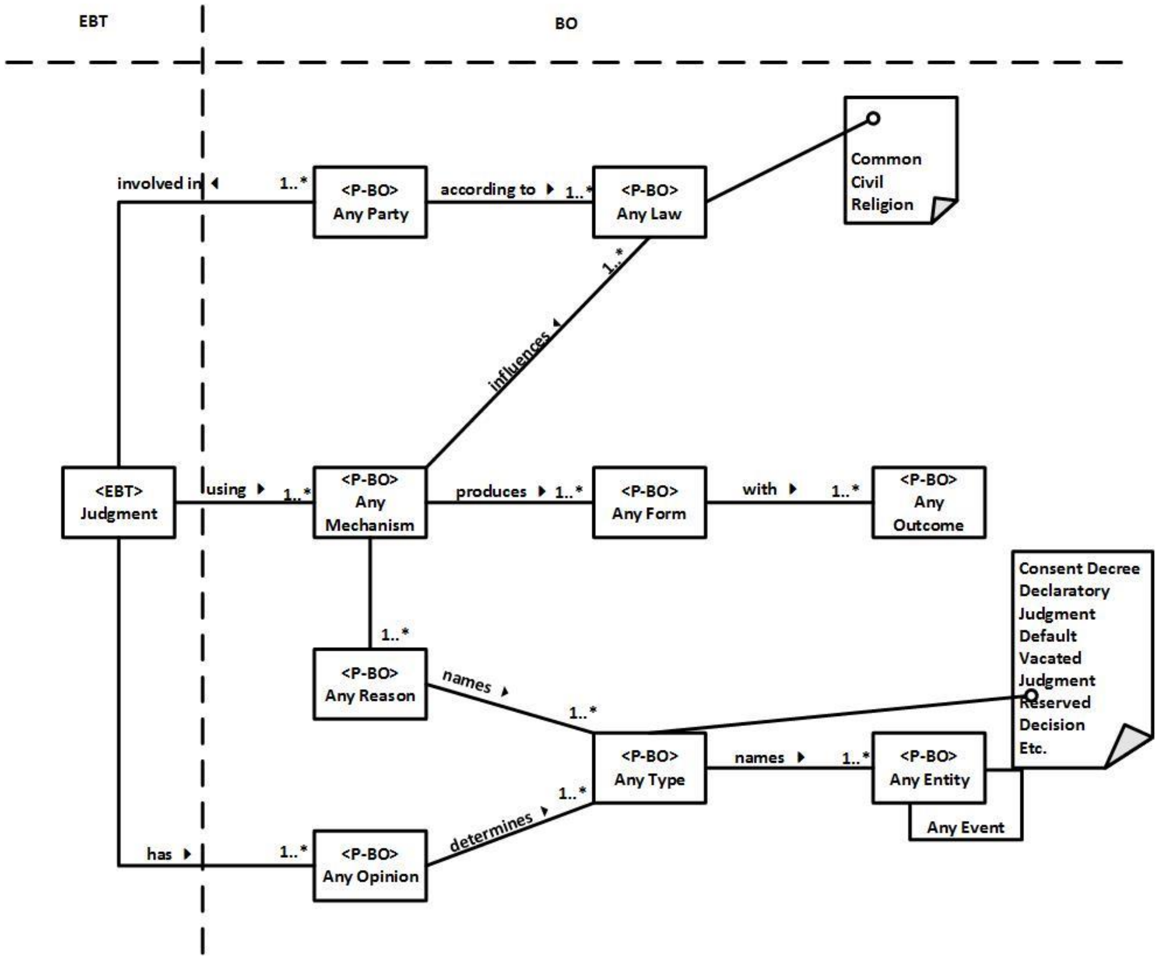


Figure 35. A class diagram for the judgment SAP.

Class diagram description.

- AnyParty is involved in AnyJudgment.
- AnyJudgment is given through AnyMechanism and has AnyOpinion.
- AnyMechanism produces AnyForm.
- AnyJudgment be according to AnyLaw.
- AnyLaw influences AnyMechanism.

- AnyReason names AnyType.
- AnyOpinion determines AnyType.
- AnyType names AnyEntity or AnyEvent.
- AnyJudgment has AnyOutcome.

Applicability. This SAP is useful in several contexts. Table 29 explores how the Judgment SAP works for several scenarios and how the classes compare in each scenario.

Table 29
Applicability of the Judgment EBT Across Several Disciplines

BO	Grades	Court Judgment	Political	Job Perform-ance	Freelancer
AnyParty	Student, Professor	Judge, Driver	Politician, Voter	Employee, Manager	Freelancer, Client
AnyLaw	Rubrics	State Law	Constitu-tional Law	Federal Laws	Code of Conduct
Any Mechanism	Reading	Settlement	Voting	Perform-ance evaluation	Online Feedback
AnyReason	Improve Perform-ance	Over Speeding	Elect Candidate	Career Develop-ment	Satisfaction
AnyEntity	Answer Sheet	-	-	-	-
AnyEvent	-	Irrespons-ible Driving	Election	Interview	Service
AnyType	Manually	Consented Judgment	Primary	Personnel	Public
AnyForm	Roster	Oral Form	Majority	Organiza-tional	Feedback Form
AnyOutcome	Grade	Penalty	Single Winner	Rating	Completion of project

AnyOpinion	Comment	Conclusion	Opinion Polls	Assess-ment	Impression

Case Study: Judgment at a court

Scenario. Consider a scenario where a driver (AnyParty) attends a court of law (AnyLaw) for speeding (AnyReason) on a freeway. The judge (AnyParty) orders a settlement (AnyMechanism) and gives the judgment in an oral form (AnyForm). The driver is fined (AnyOutcome) for speeding. The judge gives the judgment (AnyType), and the driver agrees to it, thereby providing a conclusion (AnyOpinion) for the irresponsible driving (AnyEvent). Figure 35 provides the class diagram of this case study.

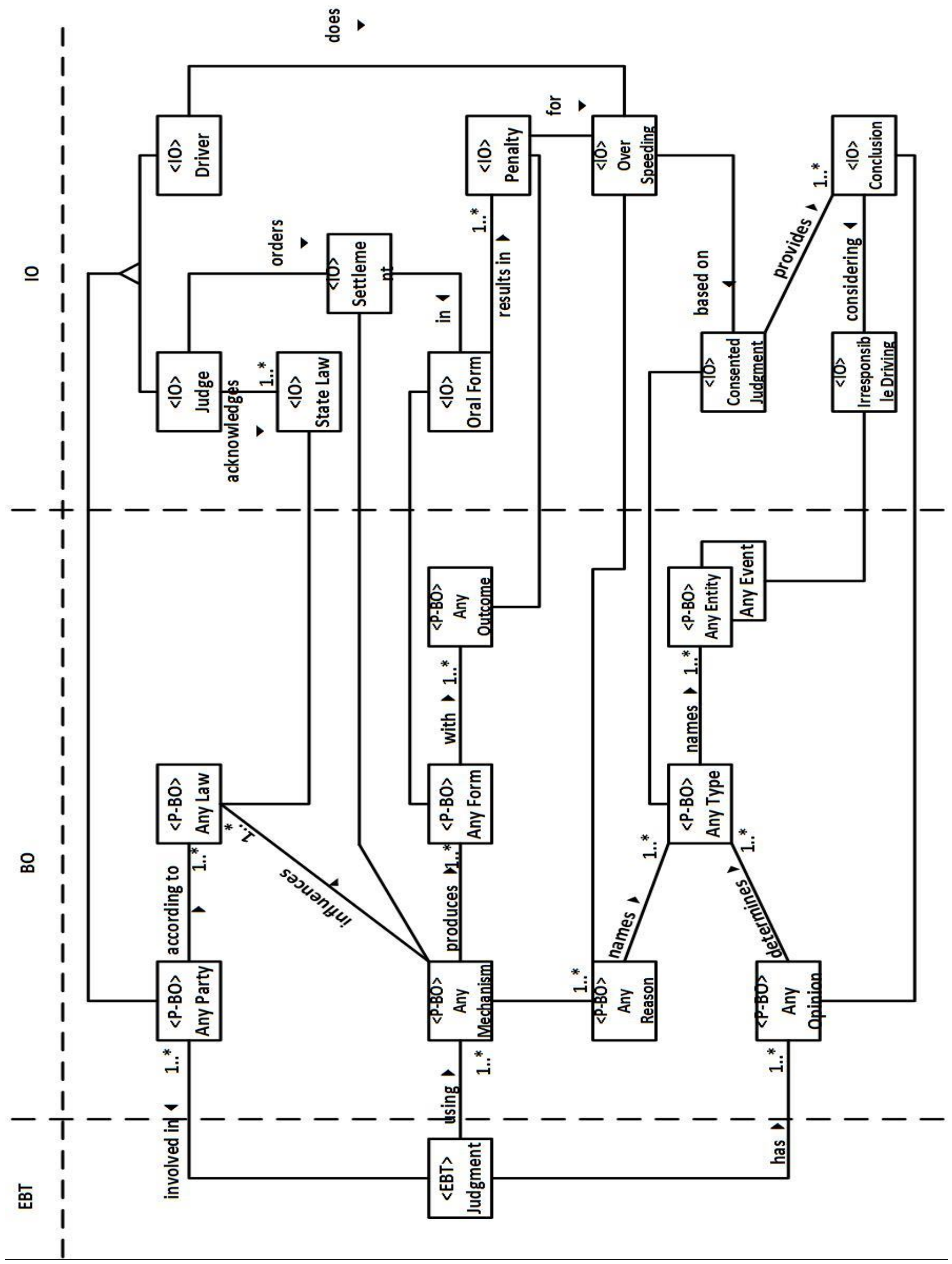


Figure 36. The class diagram for judging a traffic violation using EBTs, BOs, and IOs.

Use case and description. Consider the use case for judging a traffic violation in Table 30.

- Use Case ID: 001
- Use Case Name: Judging a Traffic Violation

Table 30
Use Case 1 for Judging a Traffic Violation

Actor	Role
AnyParty	Judge, driver

Class	Type	Attributes	Operations
Judgment()	EBT	domain, name, context, id, state	findGuilty()
AnyParty()	BO	name, id, type, phoneNumber, address	participate()
AnyLaw()	BO	number, name, type, description, regulation, associatedCharge	impact()
AnyMechanism()	BO	id, description, status, product, statement	produce()
AnyForm()	BO	number, description, name, proof, limit, case	collect()
AnyEntity	BO	name, id, type, position, state, status, isAlive	names()
AnyEvent()	BO	name, id, type, occasion, contract, contender	occur()
AnyType()	BO	id, property, subtype, interfaceList, methodList,	classify()

		name	
AnyOutcome()	BO	securityLevel, name, sector, mode, tool	complete()
AnyOpinion()	BO	limitation, opinionAbout, reference, context, description	assess()
AnyParty(Judge)	IO	name, designation, address, id, state	acknowledge()
AnyParty(Driver)	IO	licenceNumber, name, address, height, hairColor, vehicleNumber, caseNumber, chargeId	drive()
AnyLaw(State Law)	IO	stateName, country, senator, number description, associatedCharge, type	followedBy()
AnyMechanism(Settlement)	IO	id, name, type, government, regulation	happen()
AnyForm(Oral From)	IO	number, description, status, product, statement	resultsIn()
AnyEvent(Irresponsible Driving)	IO	name, id, type, position, state, status, isAlive	leadsTo()
AnyType(Consented Judgment)	IO	id, property, subtype, interfaceList, methodList, type	provide()
AnyOutcome(Penalty)	IO	reasonForPenality, name, category, mode, tool, duration, costOfPenalty	punish()
AnyOpinion(Conclusion)	IO	limitation, conclusionAbout, reference, context,	finish()

description

Use case description.

- Judgment finds AnyParty() guilty. (TC: Who finds AnyParty() guilty? Does the judgment bring any awareness?) A driver is found guilty of speeding.
- AnyParty() participates in the Judgment. (TC: Who participates in the Judgment?) A judge gives the judgment.
- AnyLaw() influences AnyMechanism(). (TC: What influences AnyMechanism? Are laws formed by the government?) The judge acknowledges state law.
- AnyMechanism() produces AnyForm(). (TC: What produces AnyForm?) The judgment is done in oral form.
- AnyForm() assembles AnyOutcome(). (TC: What assembles AnyOutcome()? Does every form have a shape and an arrangement?) The judge penalizes the accused by giving the judgment.
- AnyEntity() is named by AnyType(). (TC: What does AnyEntity() name?) The driver drives irresponsibly.
- AnyType() classifies the judgment. (How is the judgment classified? A consented judgment provides conclusion.
- AnyOutcome() completes a judgment. (What completes a judgment? What does an outcome conclude?) The penalty is given for speeding.
- AnyOpinion() assess() a judgment. (What does AnyOpinion() assess? Can belief be considered an opinion?) The conclusion is the assessment of the judgment.

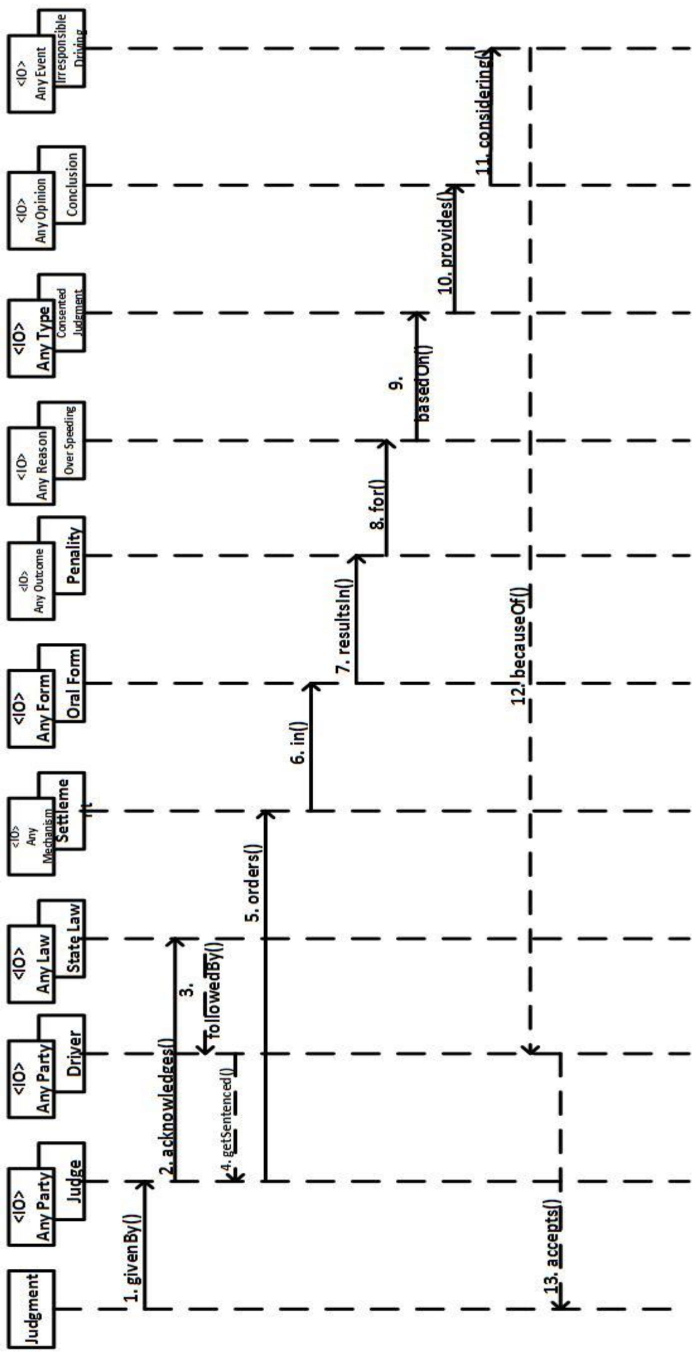


Figure 37. The sequence diagram for judging a traffic violation.

Summary. The mid-size template provided in this thesis exhibits how the core knowledge of judgment can be used for numerous applications. It interprets the SAP for judgment in an effective manner. This model can also be extended to identical contexts. Reusability of the pattern itself is a very big contribution. Due to the generality of the term *judgment*, this template needed a deeper introspection and evaluation to arrive at an EBT that reflected the overall goal of the pattern.

Need SAP

Pattern name. Need is an EBT. It is a necessary duty. Need is a demand, requirement, compulsion, or obligation. Other words for need are want, right, duty, and condition ("Need," 2016). Generality is the main reason for choosing this term because it is appropriate for all the possible scenarios of need. This leads to an SAP of need.

Context. Need has applications in different domains such as business, education, medicine, and sports. Some types of need are consent need, declaratory need, default need, divine need, and investigative need. The following are some of the contexts where we can apply need as an SAP,

Consider a scenario where a startup software organization needs to complete a project. The manager (AnyParty) assigned to this project set a due date (AnyConstraint) for its completion after discussing it with his employees (AnyParty). Startups have storage (AnyResource) as a resource within their computers, and most of the startup companies fall under the Capability Maturity Model Integration level 2 (AnyLevel) category. In order to divide the work into phases, a development methodology such as the software development life cycle (AnyMechanism) can be used to design a business plan (AnyEvent). The client demands (AnyType) the software be made, and all the employees work on-site (AnyReason).

Every human being has certain basic needs (Need) like food, water, clothing, and shelter for survival (AnyReason). Consider a situation where a father (AnyActor) provides basic necessities (AnyEntity) to his child (AnyActor), by working hard and earning money (AnyResource). Their family would be classified as a middle (AnyLevel) class family. Lack of high income (AnyConstraint) would be a hindrance to the family for satisfying their other luxury (AnyType) needs of having a bigger house, fancy car, etc. Here, we are discussing the needs of a family (AnyReason) as a whole while ignoring the individual needs of family members. Also, a father would be protective toward his child and use defensive mechanisms (AnyMechanism).

Problem. Alternatively, there is no SAP that exists for need, which defines the core knowledge of a need. Once we have an SAP for need, then we can build as many applications as we want simply by making the pattern reusable without changing the main core. Below are some of the functional and non-functional requirements:

Functional requirements.

- Need represents a necessary duty. Need has attributes such as context, type, and level. This class has operations such as satisfy(), demand(), and require().
- AnyParty represents a party that is involved in a violation. A party can be a country, political party, organization, or a person affiliated with an organization. AnyParty has attributes such as name, id, and location. It has operations such as participate(), collectData(), and interact()

- AnyActor represents someone or something that is involved in need. It has attributes such as name, id, and type. It has operations such as desire(), wishFor(), and supply().
- AnyConstraint represents something that prevents satisfying a need. It has attributes such as description, name, and id. The operations of criteria can be prevent(), test(), and measure().
- AnyMechanism represents the method used for need. It has attributes such as name, context, description, application, and status. It has operations such as execute(), attach(), detach(), and activate().
- Every need has AnyReason, and this class represents the reason for the existence of the need. AnyReason has attributes such as description, proof, and justification. It has operations such as conclude(), examine(), and resolve().
- AnyType represents the type of need. AnyType has attributes such as id, name, interfaceList, methodList, and property. The operations are change(), categorize(), and subtype().
- AnyReason helps to understand a situation completely. It has attributes such as description, scope, and category. It has operations such as validateMethodology(), analyzeCost(), and recommend().
- AnyLevel represents the degree of need. It has attributes such as type, context, and name, and operations such as balance(), equalize(), and categorize().

- AnyResource represents a supply, which one can use when needed. It has attributes such as name, id, and type. It has operations, such as supply(), provideMaterials(), and achieve().
- Any Need is done for AnyEntity. AnyEntiy named by AnyType of need. It has attributes such as id, name, and status. Its operations can be to update(), relationship(), and type().
- AnyNeed is met for AnyEvent and is named by AnyType of need. It has attributes such as name, occasion, type, and outcome. Its operations can be to appear(), happen(), and arrange().

Non-functional requirements.

- A need should have a specific limit and should be controllable. Even the basic necessities such as food and water ought to be controlled and not wasted. Consider a situation where a person who was very much looking forward to buying a new car, and Finally, buys a car. After a few months, he is bored with his car and wants to buy another one, when there is no need at all to buy another car.
- A need should be essential in nature for something to happen. Consider a scenario where a student plans to buy a laptop. The student has two options: buying a new laptop or a refurbished version of the same laptop for a lesser price. Buying a laptop could be a necessity for the student because he needs a laptop to complete his class assignments, but whether he needs a new laptop or a refurbished laptop to meet his needs is difficult to answer. In another instance, a

homeless person needs food to survive; otherwise, he or she could die due to hunger. Here, the need for food is necessary, or it causes the death of a homeless person. Therefore, every need should be defined according to its necessity.

- Consider a scenario where a person needs to affiliate with something. There is an organization for youth worldwide to put forward their leadership abilities. A student may have a tendency and need to get involved in such an organization. But, he is not qualified for such a huge organization, so he opts for a small college club depending on his existing capabilities. Need should be measurable based on various factors. Some needs are big and unrealistic, whereas some needs are achievable, such as in the case of the homeless person who needs a home to survive. He or she cannot look at a big house because he or she is not capable of buying the big house. However, it is possible for a homeless person to buy a small house after a long course of hard work. Therefore, a need should be measurable.
- A need should be fulfilled on time. If there is a natural disaster like floods in a village and there is a need for food, clothing and shelter for the affected people, then it makes no sense if someone donated clothes several months after the disaster has occurred. Every need has a stipulated time and is most effective when fulfilled on time.
- Available needs should be adequate and sufficient. Needs should not be too great, and if there is excess available, then the person may not need anything more. For example, a person who is facing a persistent financial problem may just need

sufficient food, clothing, and housing. However, if the same person gets an adequate amount of food, clothing, and housing, then he or she may need more money to improve the quality of those things. Similarly, a less intelligent student who always scores lower marks may just need passing grades to graduate whereas an intelligent student always seeks the highest grades. Thus, a need is an ever-growing concept where one always craves for more and more. A need should always be adequate enough, only to fulfill the requirements.

Solution. This paper provides a workable solution by utilizing the basic concepts of the SSM to extract the core knowledge of need. It also focuses on how a need be given. The given solution also provides an SAP that incorporates various methods to apply need and to build numerous applications on cross-platform domains. Figure 37 shows the class diagram of Need SAP.

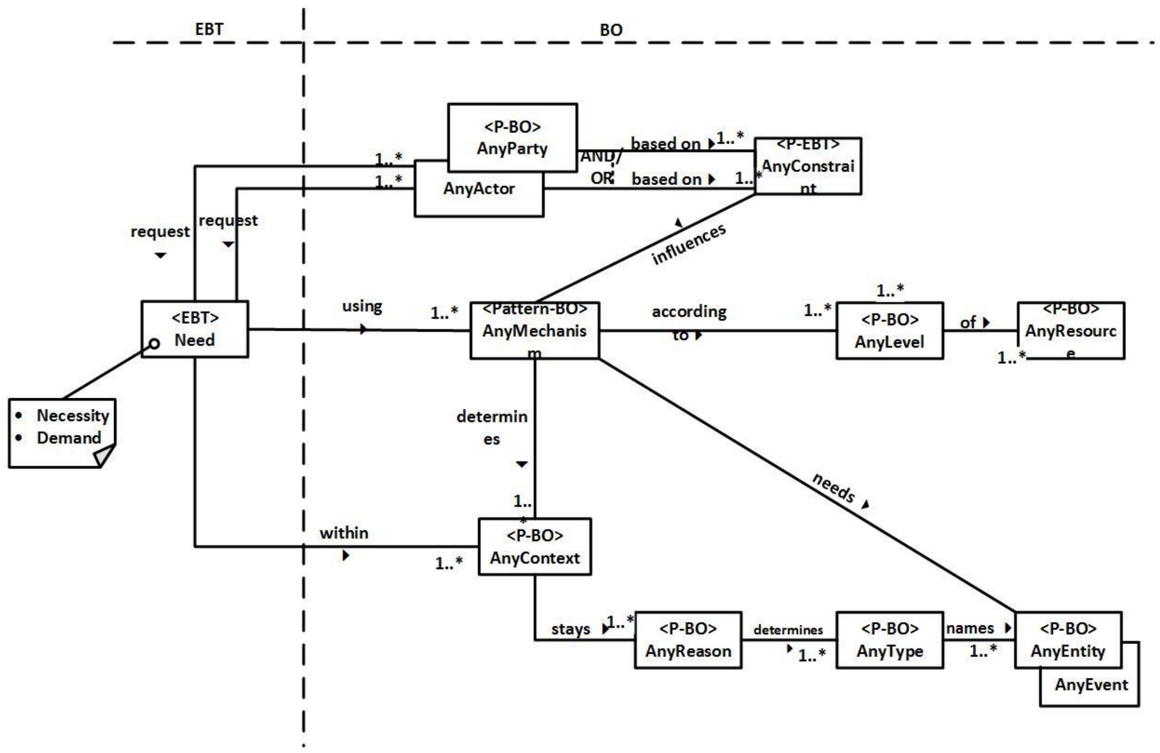


Figure 38. Diagram of the Need SAP.

Class diagram description.

- AnyParty or AnyActor request AnyNeed based on AnyConstraint.
- AnyNeed exists using AnyMechanism within AnyReason.
- AnyConstraint influences AnyMechanism.
- AnyMechanism determines AnyReason according to AnyLevel.
- AnyReason determines AnyType.
- AnyType names AnyEntity or AnyEvent.
- AnyEntity or AnyEvent is needed by AnyMechanism

Applicability. This SAP is useful in several contexts. Table 31 explores how the Need SAP could benefit several situations and how the classes compare in each scenario.

Table 31
Applicability of the Need EBT Across Several Disciplines

BO	Organ-izational Needs	Psychological Needs	Financial Needs	Insurance Needs	Sugical Needs
AnyParty	Manager	-	Lender	Insurance Company	Surgeon
AnyActor	Employee	Father, Son	Student	Insurance Holder	Patient
Any Mechanism	Office Work	Hard Work	Credit	Policy	Operation
AnyReason	Client Demand	Survival	School Fee	Illness	Skin Care
AnyEntity	-	Basic Necessities	-	-	-
AnyEvent	Business Plan	-	Studying	Accident	Medical Service
AnyType	Software	Whole	Secured Loan	Life Insurance	Cosmetic
Any Constraint	Project Deadline	High Income	Payment Dates	Legal Contract	Aging
AnyLevel	Bench-mark	Middle	Credit Level	Family Protec-tion	Skin Grade
AnyResource	Computer, reusable parts, tools	Money	Bank	Invest-ment	Hospital
AnyReason	Onsite chance	Family	Education	Coverage	Medicine

Case Study: Organizational needs

Scenario. Consider a scenario where a manager (AnyParty) to whom a project is assigned sets a project deadline (AnyConstraint) after discussing the project with his or her employees (AnyParty). Employees create a benchmark (AnyLevel) of their work by using computers (AnyResource) and reusable components or tools (AnyResource) as their resources. The employees work at an office (AnyMechanism) and design the business plan (AnyEvent). The client demands (AnyReason) software (AnyType) and promises all the employees to relocate them onsite (AnyReason) when the software is successfully delivered. Figure 37 provides the class diagram of this case study.

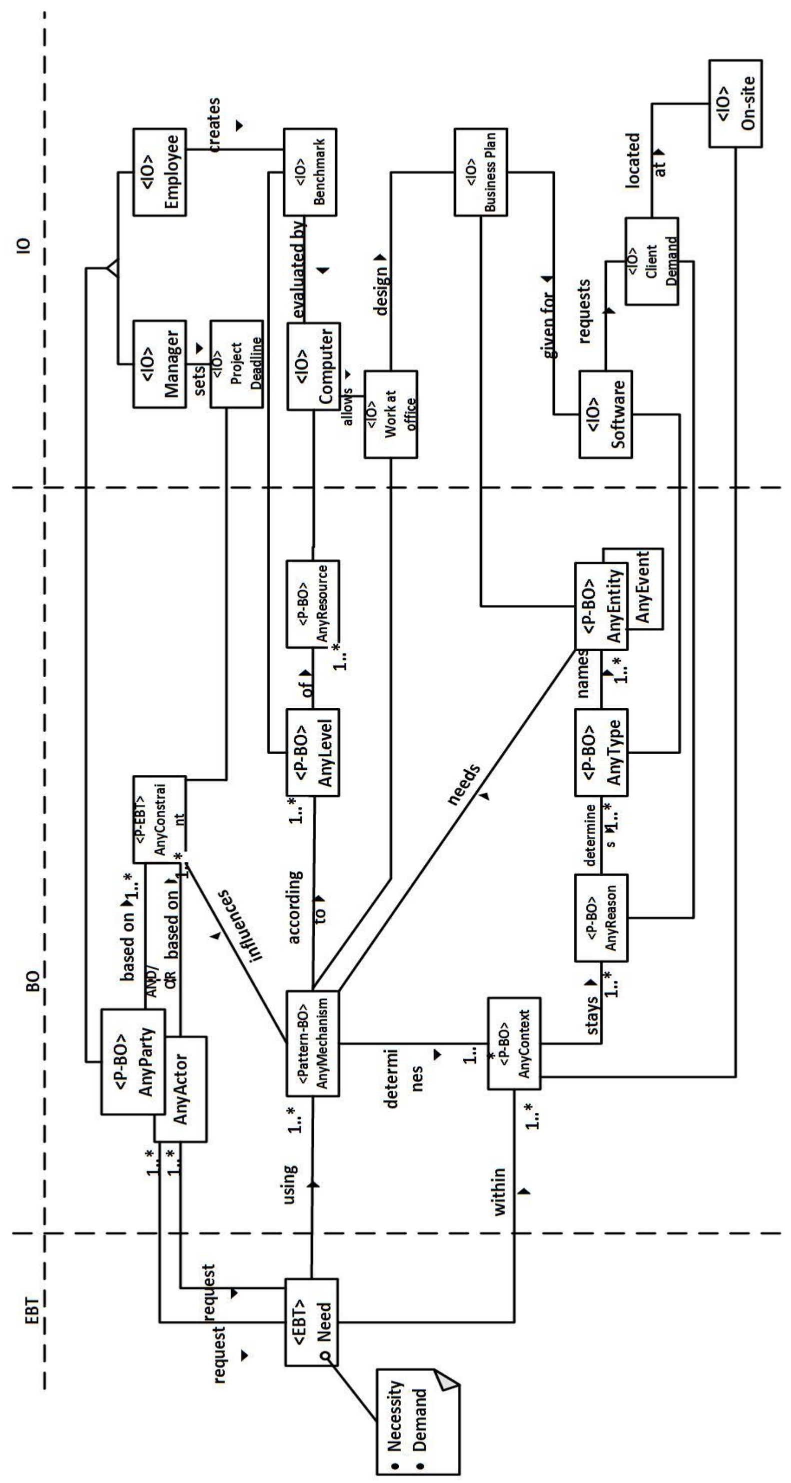


Figure 39. The class diagram for assessing the needs of an organization.

Use case and description. Consider the use case for assessing organizational needs in Table 32.

- Use Case ID: 001
- Use Case Name: Returning a Lost Item

Table 32
Use Case 1 for AnyNeed: Returning a Lost Item

Actor	Role
AnyParty	Manager, employee

Class	Type	Attributes	Operations
Need()	EBT	typeOfNeed, domain, name, context, id, state	require()
AnyParty()	BO	name, id, type, phoneNumber, address	participate()
AnyActor()	BO	id, name, type, government, regulation	furnish()
AnyMechanism()	BO	name, description, status, product, statement	satisfy()
AnyConstraint()	BO	id, description, name, proof, limit, case	impact()
AnyEntity()	BO	name, id, type, position, state, status, isAlive	names()
AnyEvent()	BO	name, id, type, occasion, contract, contender	occur()
AnyType()	BO	id, property, subtype, interfaceList, methodList, name	classify()

AnyReason()	BO	description, proof, justification, name, explanation	determine()
AnyReason()	BO	limitation, contextID, reference, context, description	check()
AnyLevel()	BO	securityLevel, name, sector, mode, tool, severity	refer()
AnyResource()	BO	name, id, category, description, usedFor	supply()
AnyParty(Manager)	IO	department, name, address, phoneNumber, email	sets()
AnyParty(Employee)	IO	employeeNumber, name, address, height, hairColor	create()
AnyMechanism(Work at Office)	IO	id, name, status, application, description, context	design()
AnyConstraint(Project Deadline)	IO	restrictionName, type, severity, threshold, description	control()
AnyEvent(Business Plan)	IO	objective, policy, strategy, income, position	summarize()
AnyType(Software)	IO	code, instruction, systemName, application, data	request()
AnyReason(Client Demand)	IO	name, id, type, position, state, status, softwareName	locatedAt(), findRequirement()

AnyReason(On-site)	IO	id, property, name, website, livingCost	offeredTo()
AnyLevel(Benchmark)	IO	standard, quality, regulation, experience, legal	evaluatedBy()
AnyResource(Computer)	IO	factor, means, capital, information, activity	allow()

- There are several interactions and qualifiers that determine the use case. Some of these are as follows.Need() is a requirement of something requested by AnyParty(). (TC: Who requests AnyNeed?) The manager needs his employees to work hard.
- AnyParty() participates in Need. (TC: Who participates in AnyNeed?) An employee works according to the needs of the manager.
- AnyMechanism() satisfies Need(). (TC: What satisfies Need()? What does AnyMechanism() compensate?) Computers, reusable components, and tools allow employees to work at an office.
- AnyConstraint() influences AnyMechanism(). (TC: What influences AnyMechanism()?) Project deadlines influence the work at the office.
- AnyEvent() is named by AnyType(). (TC: What does AnyEvent) name? When does the event occur?) A business plan is given for the software creation.
- AnyType() classifies a need. (How is a need classified?) A client demands software.

- AnyReason() determines AnyType(). (What determines AnyType()?) The client’s demands determine the software and provide an on-site opportunity for the employees.
- AnyLevel() refers to AnyResource(). (TC: What does AnyLevel() refer to? What does a level qualify?) The computer and other tools evaluate a benchmark.
- AnyResource() supplies something to the need. (TC: What is supplied to need?) The computer and other tools help the employees to work.
- AnyReason() assesses a need. (What does AnyReason() assess? Which situation does a reason assess?) The onsite location opportunity assesses the need for software.

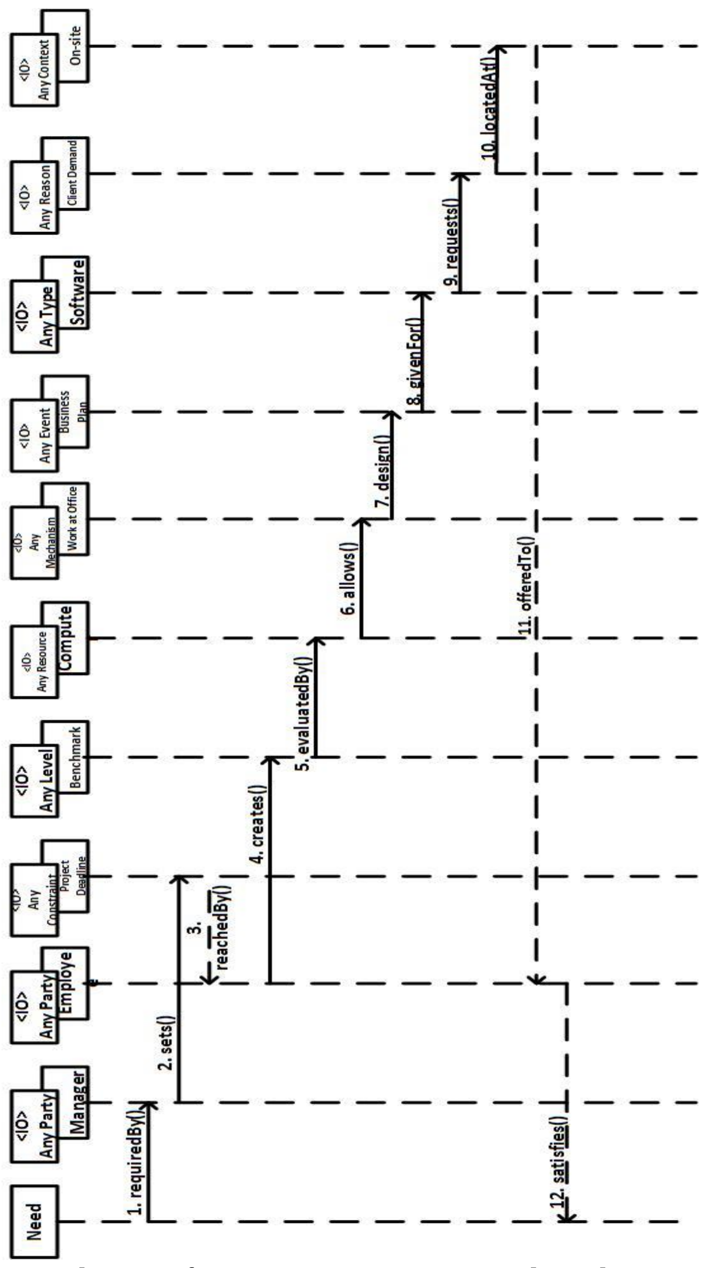


Figure 40: A sequence diagram for assessing organizational needs.

Summary. The mid-size template provided in this paper exhibits how core knowledge of need can be used for numerous applications. Being a general term in nature, a need pattern template is possible when we identify the core theme for the word and use it to create a solid and workable template. It interprets SAP for need in an effective manner. This model can also be extended to identical contexts. Reusability of the pattern itself is a very big contribution.

Ownership SAP

Pattern name. Ownership is an EBT. It means having ultimate rights over something that can be claimed lawfully. Ownership is to acquire, occupy, maintain, dominate or enjoy something. Other words for ownership are holding, takeover, possession, purchase or partnership ("Ownership," 2016). Generality is the main reason for choosing this term because it is appropriate for all the possible scenarios of ownership. This lead to an SAP of ownership.

Context. Ownership has numerous applications in different domains such as business, education, medicine and sports. Some types of ownership are consent ownership, declaratory ownership, default ownership, divine ownership, and investigative ownership. The following are some of the contexts where we can apply ownership as an SAP.

Consider a scenario where a landlord posses ownership of a land (AnyParty). The landlord leases land to a tenant (AnyParty) hoping for income capitalization (AnyMechanism). The landlord has the land registration documents (AnyEvidence) as a proof of his ownership, and he or she expects some capital gain (AnyGain) from this process by applying real estate financial modeling technique. The landlord may choose to display a miniature house (AnyModel) to prospective tenants instead of taking them to the site. The tenant would have the right of possession (AnyRight) to the land until he leases the estate (AnyEntitiy). This type of ownership is known as leasehold (AnyType) estate.

Or, consider that we all use vehicles for our daily transportation. Many of us own a vehicle (AnyEntity), and someone may want to transfer the ownership for some reason. Consider a situation where a car dealer (AnyParty) transfers the ownership of a car to a buyer (AnyParty). The dealer has the title (AnyEvidence) for the car that establishes him as the legal owner of the car. The dealer sells the car for a reasonable annual percentage rate (AnyGain), depending on the make and year model (AnyModel) of the car. The buyer can demand a smog inspection (AnyRight) because a vehicle that fails smog testing is considered unfit for driving. The buyer trades in (AnyMechanism) his old vehicle when buying a new one. He asks the dealer to reduce the price of the new (AnyType) vehicle accordingly.

Problem. There is no SAP existing for ownership, which defines the core knowledge of an ownership. Once we have an SAP for ownership, then we can build many applications as many applicationsas we want, by using it repeatedly, eventually making the pattern reusable without changing the main core. Let us evaluate some of the functional and non-functional requirements.

Functional requirements.

- Ownership represents a necessary duty. Ownership has attributes such as context, type, and level. It has operations such as satisfy(), demand(), and require().

- AnyParty represents a party that is involved in the violation. A party can be a country, political party, organization, or a person affiliated with an organization. AnyParty has attributes such as name, id, and location. It has operations such as participate(), collectData() and interact()
- AnyActor represents someone or something that is involved in ownership. It has attributes such as name, id, and type. It has operations such as desire(), wishFor() ,and supply().
- AnyConstraint represents something that prevents satisfying an ownership. It has attributes such as description, name, and id. The operations of criteria can be prevent(), test(), and measure().
- AnyMechanism represents the method used for ownership. It has attributes such as name, context, description, application, and status. It has operations such as execute(), attach(), detach(), and activate().
- Ownership has AnyReason and this class represents the reason for the existence of ownership. It has attributes such as description, proof, and justification. It has operations such as conclude(), examine(), and resolve().
- AnyType represents the type of ownership. AnyType has attributes such as id, name, interfaceList, methodList, and property. The operations are change(), categorize(), and subtype().
- AnyReason helps in understanding a situation easily. It has attributes such as description, scope, and category. It has operations such as validateMethodology(), analyzeCost(). and recommend().

- AnyLevel represents the degree of ownership. It has attributes such as type, context, and name. This class has operations such as balance(), equalize(), and categorize().
- AnyResource represents a supply, which one can use when owned. It has attributes such as name, id, and type. It has operations such as supply(), provideMaterials(), and achieve().
- AnyEntity takes ownership of something and is named by AnyType of ownership. It has attributes such as id, name, and status. Its operations can be to update(), relationship(), and type().
- Ownership is required for AnyEvent and is named by AnyType of ownership. It has attributes such as name, occasion, type, and outcome. Its operations can be to appear(), happen(), and arrange().

Non-functional requirements.

- Ownership should have a limit and be controllable. A person who owns something through a legal transaction should have complete control over buying other things. As AnyActor makes a payment, he or she has the freedom to exercise control over the purchase. For instance, if a customer buys a car from a seller and pays the full amount to the seller, it does not mean that buyer has a complete control (ownership) of the car. He or she has to transfer the legal documents for the car into his or her name. Thus, control over the things, product, or article is necessary for exercising complete ownership.

- Ownership should be essential in nature for something to happen. Consider a scenario when a person wants to own an apartment. The ownership is need-based. A person can only buy and own something that is necessary to satisfy an immediate need. An example is buying and owning a car. Cars come in many shapes, sizes, and costs. A person should buy and own only those cars that are necessary and within his or her budget. There should always be necessity attached to the ownership; otherwise, ownership won't have any limitation.
- Ownership of any thing or object should be measurable regarding its monetary value and quality. Considering the similar example in the previous section, the ownership of car or house can only be measured once the cost of the car or house is definite. The buyer who wants to buy a car should have an adequate amount of money in order to transfer the ownership of the car. Here, ownership of the car is measurable, and it is measured by the cost of the car.
- An ownership transaction should be carried out on time. If the ownership is not transferred on time, there can be a huge loss to the person who buys something. For example, in an escalating real estate market, the buyer needs to be active to buy the property because the price of any property changes rapidly. So, if a buyer misses one day, then he may have to pay more money the next day due to a rise in the price of a property. Similarly, in the case of the stock market, ownership of shares gets transferred within minutes due to the fluctuating behavior of the stock market.

Solution. This chapter provides an enduring solution by utilizing the principles of SSM to extract the core knowledge of ownership. It also focuses on how an ownership needs to be given. The given solution provides an SAP that incorporates various methods to apply ownership and to build numerous applications on cross-platform domains. Figure 38 shows the class diagram of Ownership SAP.

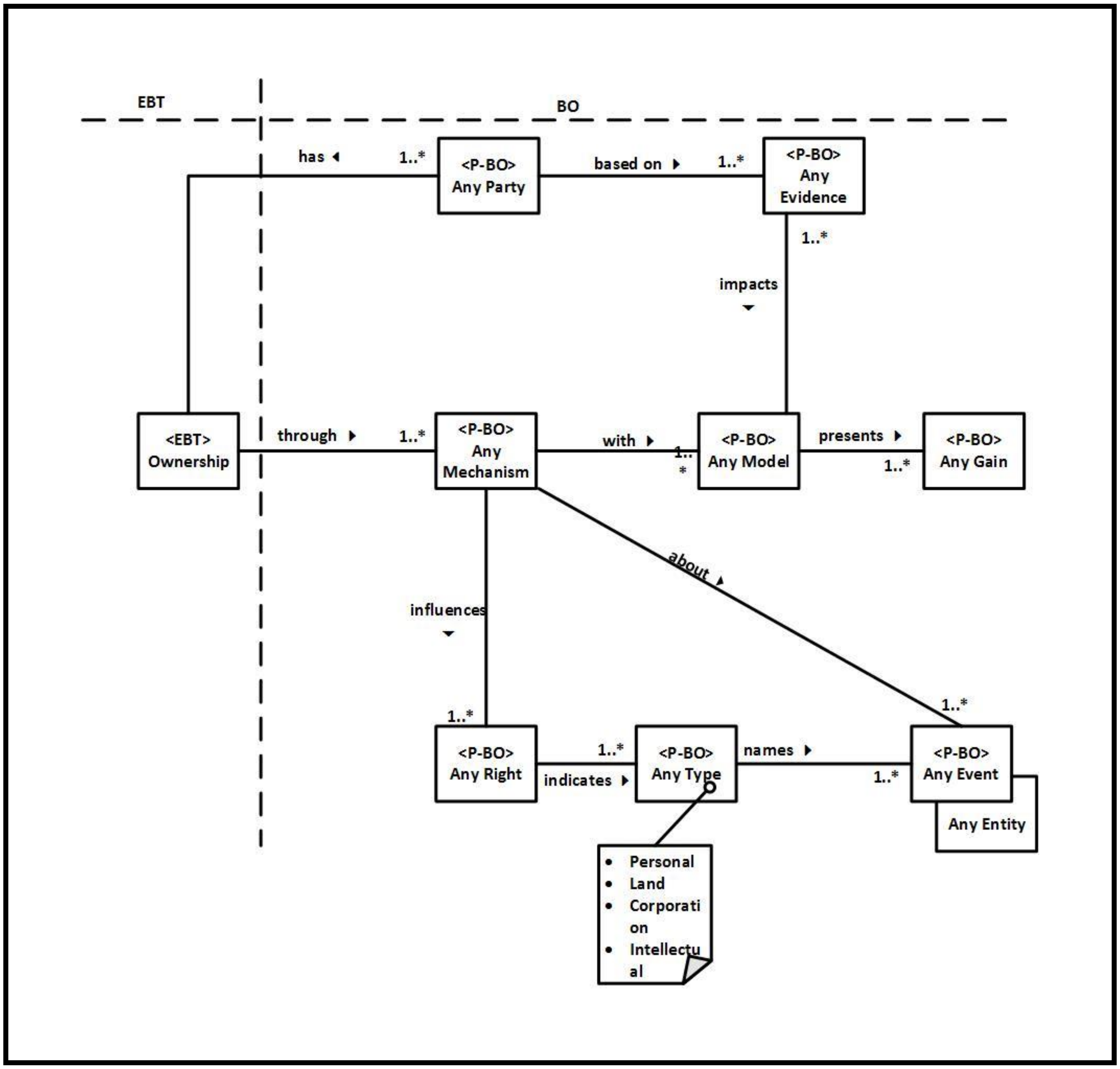


Figure 41. A diagram of an ownership SAP.

Class diagram description.

- AnyParty or AnyActor request ownership based on AnyConstraint.
- Ownership exists using AnyMechanism for AnyReason.
- AnyConstraint influences AnyMechanism.
- AnyMechanism determines AnyReason according to AnyLevel.
- AnyContext stays for AnyReason.
- AnyReason determines AnyType.
- AnyType names AnyEntity or AnyEvent.
- AnyEntity or AnyEvent occurs through AnyMechanism

Applicability. This SAP is useful in several contexts. Table 33 explores how the Ownership SAP could work across several scenarios and how the classes compare in each situation.

Table 33
Applicability of the Ownership EBT Across Several Disciplines

BO	Land	Vehicle	Work	Medical Records	Video Footage
AnyParty	Land-lord, Tenant	Buyer, Seller	Employee, Employer	Patient, Physician	Ad agency, Client

AnyEvidence	Regis-tration Docu-ments	Pink slip	Work Statement	Private Practice	Video
AnyMechanism	Income Capitalization	Deal	Top-bottom approach	Tool	Shoot
AnyGain	Capital Gain	Money	Salary	Patient Responsibility	Views
AnyEntity	Estate	Vehicle	Project	Records	Unused Footage
AnyEvent	-	-	-	-	-
AnyType	Lease-hold	Exclusive	Account-ability	Medical	Ad-campaign
AnyModel	Financial Model	Year Model	Waterfall	Medical Board Regulations	Third Party
AnyRight	Right of Possession	Smog Inspection	Right to Make Decisions	Right to Inspect	Legal Rights
AnyConstraint	DNR order	Noise	Number of students	Neglect	30 days

Case Study: Organizational ownerships

Consider a scenario where a landlord (AnyParty) possesses a piece of land and plans to rent it for profit. The land registration documents (AnyEvidence) serve as evidence of the ownership and can be reviewed by the tenant (AnyParty) before renting the property. Also, the landlord validates the capital gain (AnyGain) by following a financial model (AnyModel) to maintain income capitalization (AnyMechanism) and to identify the leasehold (AnyType). The tenant has the right of possession (AnyRight) of the estate until the lease ends. Figure 39 provides the class diagram of this case study.

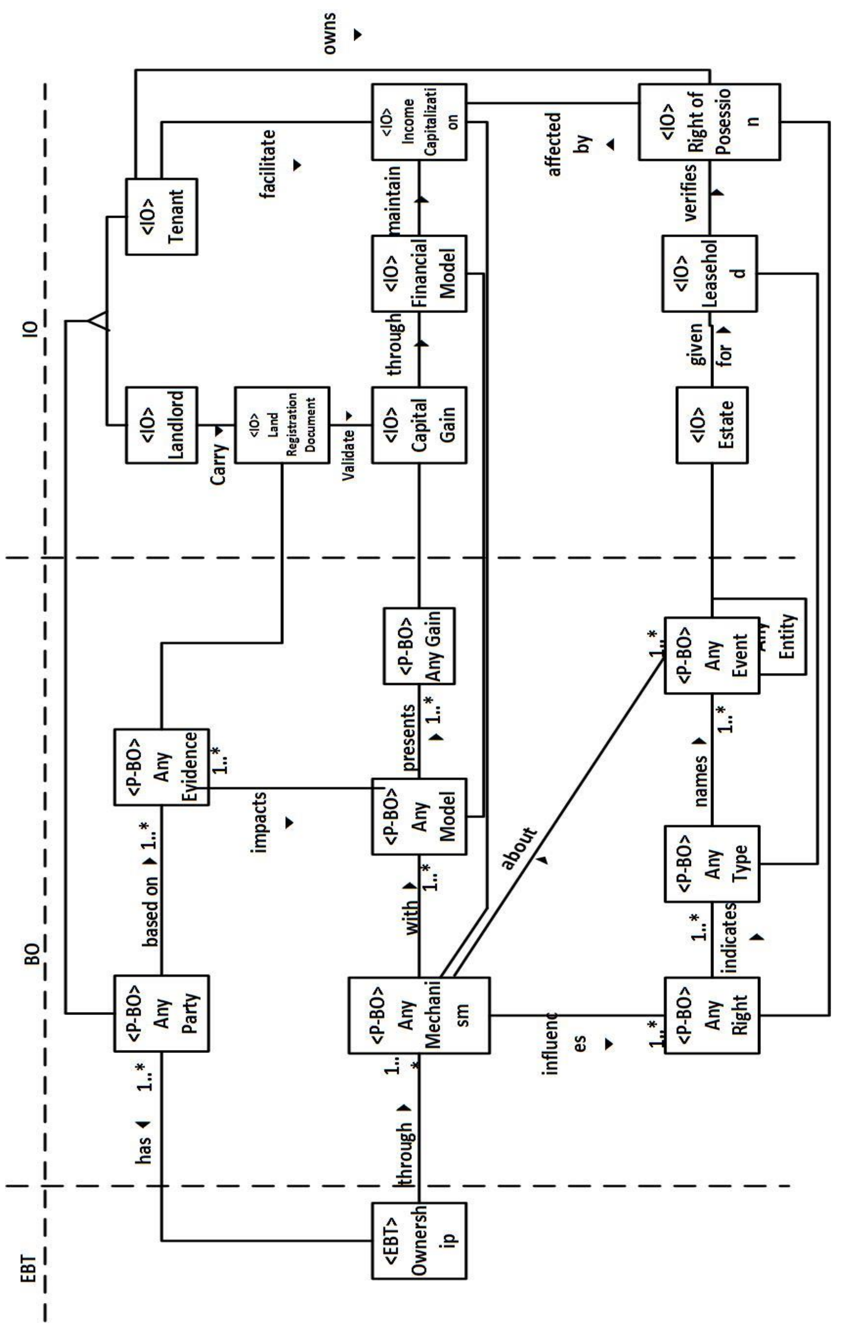


Figure 42. The class diagram for assessing the ownership of an organization.

Use case and description. Consider the use case for assessing organizational ownership in Table 34.

- Use Case ID: 001
- Use Case Name: Assess Organizational Ownerships

Table 34

Use Case 1 for AnyOwnership

Actor	Role
AnyParty	Landlord, tenant

Class	Type	Attributes	Operations
Ownership()	EBT	domain, name, context, id, state	possess()
AnyParty()	BO	name, id, type, phoneNumber, address	participate()
AnyMechanism()	BO	id, description, status, product, statement, application	accomplish()
AnyEvidence()	BO	id, description, name, proof, limit, case, typeOfEvidence	convince()
AnyEntity()	BO	name, id, type, position, state, status, isAlive	names()
AnyEvent()	BO	name, id, type, occasion, contract contender, isActive	occur()
AnyType()	BO	id, property, subtype, interfaceList, methodList, name	classify()
AnyModel()	BO	limitation, id, reference, context,	design()

		description, application	
AnyGain()	BO	level, name, sector, mode, tool	achieve()
AnyRight()	BO	name, id, category, description, usedFor	express()
AnyParty(Landlord)	IO	propertyType, name, address, phoneNumber, email	carry()
AnyParty(Tenant)	IO	ssnNumber, name, address, height, hairColor	validate()
AnyMechanism(Income Capitalization)	IO	id, name, status, application, description, context	change()
AnyEvidence(Land Registration Document)	IO	name, type, id, threshold, description	review()
AnyEntity(Estate)	IO	name, country, city, incomeposition	ownedBy()
AnyType(Leasehold)	IO	name, id, type, position, state, status	gives()
AnyModel(Financial Model)	IO	standard, quality, regulation, experience, legal	maintain()
AnyGain(Capital Gain)	IO	factor, means, capital, information	improve()
		activity	
AnyRight(Right of Possession)	IO	name, id, power, priority, duty	verify()

The use case allows designers to see the interactions and qualifiers for each class.

Below is the description for the use case.

- Ownership() is possession of something by AnyParty(). (TC: What is possessed by AnyParty()? How does ownership control something?) A landlord possesses a building.
- AnyParty() participates in Ownership. (TC: Who participates in Ownership?) The landlord has the land registration documents.
- AnyMechanism() satisfies Ownership(). (TC: What satisfies Ownership()? What are the different types of mechanism for ownership?) Income affects property possession.
- AnyEvidence() proves AnyMechanism(). (TC: What influences AnyMechanism()?) The tenant proves his income by showing the landlord paystubs.
- AnyEntity() is named by AnyType(). (TC: What does AnyEntity() name? What is the meaning of AnyEntity? Is AnyEntity an object?) The landlord owns the estate.
- AnyType() classifies the ownership. (How is the ownership classified?) The property is classified as multi-family rental property.
- AnyModel() displays AnyGain(). (What presents AnyGain()? What example does the model provide?) The financial model lists income growth from each property.
- AnyGain() achieves Ownership(). (TC: What achieves AnyGain()?) Capital gain is achieved by following a financial model.

- AnyRight() indicates Ownership() (TC: What indicates ownership? Who signifies ownership?) The title verifies the leasehold.

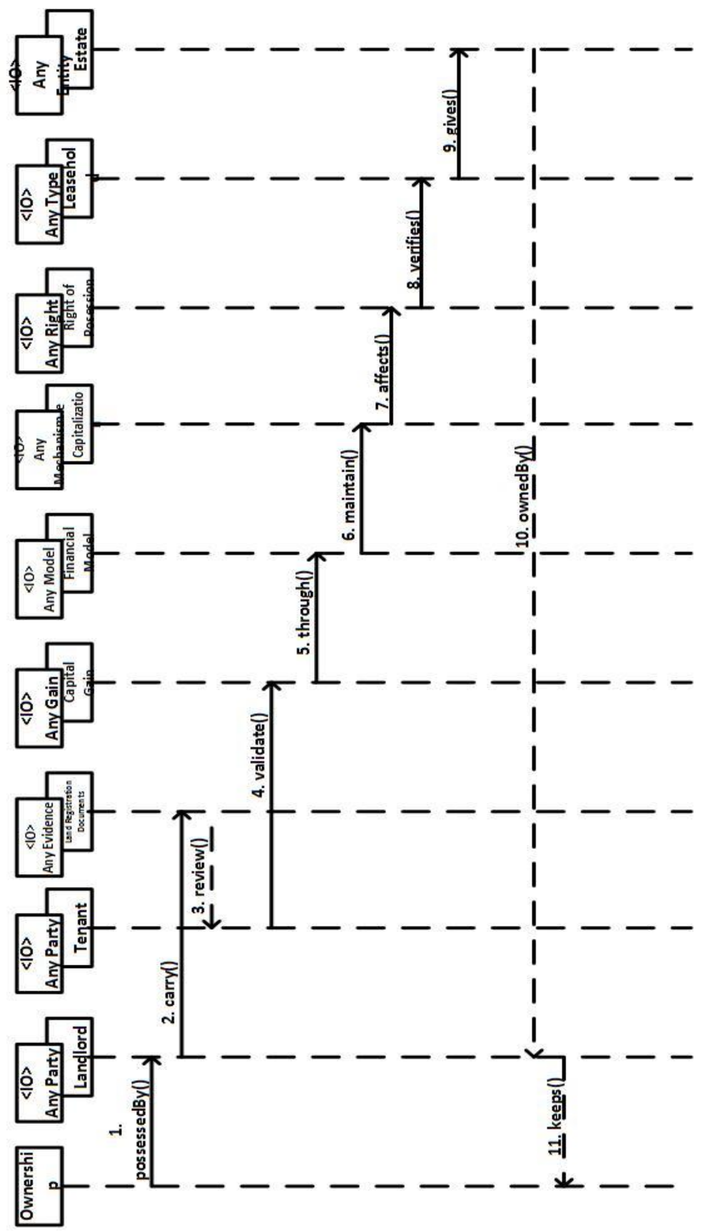


Figure 43. The sequence diagram for assessing the ownership of an organization.

Summary. The mid-sized template provided here exhibits how core knowledge of ownership can be used in numerous applications. Since the term *ownership* has varied uses in different contexts, it is important to identify an essential theme that makes up for the stable pattern. The template identified and designed in this section serves as a strong base for creating a stable pattern on ownership. It interprets the SAP for ownership in an effective manner. This model can also be extended to identical contexts.

Chapter 5: Short-size Documentation of Stable Analysis and Design Patterns

Pattern Documentation

Pattern name: AnyAppraisal SDP. Appraisal is a BO. It is the action of determining the worth of something ("Appraisal," 2016). The main reason for choosing this term is its wide-reaching nature. This term is appropriate for all possible scenarios of evaluation. This generality leads to an SDP using appraisal as its EBT.

Context. Appraisal has applications in various domains such as decision-making, property evaluation, art, and business. Some types of appraisals are project appraisals, economic appraisals, or performance appraisals. Consider the following situations where the AnyAppraisal SDP could be used.

In one situation, a set of students (AnyParty) plan to buy an apartment (AnyEntity). The leasing officer (AnyParty) lists the terms and conditions (AnyCriteria) such as the duration of the lease and the income requirements necessary for leasing the apartment. The apartment's base rent is evaluated (AnyAppraisal) based on the size of the apartment, the number of bedrooms and baths in the apartment, and the neighborhood where the apartment is located. The Internet (AnyMedia) has numerous websites that maintain databases (AnyLog) of tenant-generated reviews (AnyRating) to help prospective tenants in deciding whether to rent the apartment or not.

Another example of AnyAppraisal is a performance appraisal. This is a methodology that evaluates an employee's (AnyParty) performance at work. It is a periodic assessment (AnyEvent) with respect to a set of preestablished objectives (AnyCriteria). One common method is a judgmental evaluation (AnyAppraisal) where raters (AnyParty) such as the job supervisor or the employee's colleagues collect data. In a peer rating method, each employee ranks the other members of their group from best to worst using an online survey tool (AnyMedium). The survey responses are stored in company records (AnyLog) and used for the employee's career development.

Problem. The term *appraisal* is too general to be used for a specific domain. It has a wide array of uses and only explains the core concept of an entity that evaluates a thing, an event, an application, or service for its worth, benefit, cost, quality, or valuation.

It is important to understand how an appraisal could suit any number of software applications. The AnyAppraisal SDP consists of the core knowledge of an appraisal. It deals with its enduring theme (evaluation), which makes it stable, robust, and reusable over an unlimited number of applications. Given below are some of the functional requirements of AnyAppraisal and the quality factors that assist in better understanding the SDP of AnyAppraisal.

Functional requirements.

- Evaluation: Evaluation means to judge in a cautious way. It fixes the value of something. For instance, before a person is recruited into the army, a medical evaluation of the candidate is done to make sure that he or she is capable of performing military duties.
- AnyAppraisal represents the action of determining the importance or determining the value of something. In other words, the assessment of something is AnyAppraisal.

- AnyParty is a legal user such as an organization, a person, a government, or a political party that can do AnyAppraisal. AnyParty has a name, contact information, and at least two roles. It has operations such as participate(), playRole(), interact(), and setCriteria().
- AnyCriteria represents something that is used as the purpose of a judgment or choice. Any Criteria defines the standard. It has attributes such as description, name, and id. The operations of criteria can be judge(), prove(), and measure().
- AnyAppraisal can be done for AnyEntity such as a car dealership or school. The attributes can be id, name, and status. Its operations can be to update(), relationship(), and type().
- Any Appraisal can happen for AnyEvent such as an essay writing competition or a concert. AnyEvent has a name, a reward, and type. AnyEvent can have operations such as occur(), gather(), and involve().
- AnyMedia is the medium through which the appraisal is done. AnyAppraisal can occur through AnyMedia such as a phone call, a newsletter, or an email. The data and information about AnyAppraisal are stored on AnyLog. AnyMedia has a name, an id, and a category. A medium connects() different entities and events to each other. It allows users access() to AnyLog. AnyMedia helps to broadcast() various appraisals and display() them as well.
- The appraisals given through AnyMedia are stored in AnyLog. AnyLog allows users to search for a specific appraisal and has attributes such as id, name,

numberOfEntires, and references. Operators include stored(), search(), insert(), or extend().

- AnyRating represents a scaled assessment. Rating is frequently subjective but can be objective. For example, a television program's rating may depend on how many people it. A few of the other items that receive ratings are movies, video games, and restaurants.

Non-functional requirements.

- AnyAppraisal has to be relevant. It should be appropriate, meaningful, and significant. For example, if there is an employee who receives an evaluation from the manager, then evaluation should be relevant to the employee's role in the organization. Evaluating a janitor for his ability to file documents is not relevant.
- An appraisal should make a job easier to do by helping others. AnyAppraisal is an evaluation of something and should help others in making decisions about the product or service. If AnyAppraisal is done for a laptop, for example, its worth would is decided based on the clarity, features, and specifications. If an appraiser simply says, "I liked the laptop" it is not helpful to anyone.
- AnyAppraisal should be acceptable. When a well-known writer writes a new book, he sends the manuscript for appraisals from book editors, writers, and readers. A review from an unpublished writer is not acceptable as an appraisal.
- AnyAppraisal should teach the appraisee something new. AnyAppraisal gives opinions and supports them with facts. If there is a flaw in the service of a business, consultants should uncover the flaw and teach the business owner's

where they can improve. If every consultant points out the same flaw, there is no new discovery in their appraisal.

Solution. This paper provides a simple solution by using the principles of SSM to carve out the core knowledge of AnyComplaint. It also focuses on how to create AnyAppraisal when something needs to be appraised by AnyActor or AnyParty. The given solution provides an SDP that includes different methods to apply AnyAppraisal and to design numerous applications on cross-platform domains. Figure 41 shows the class diagram of the AnyComplaint SDP.

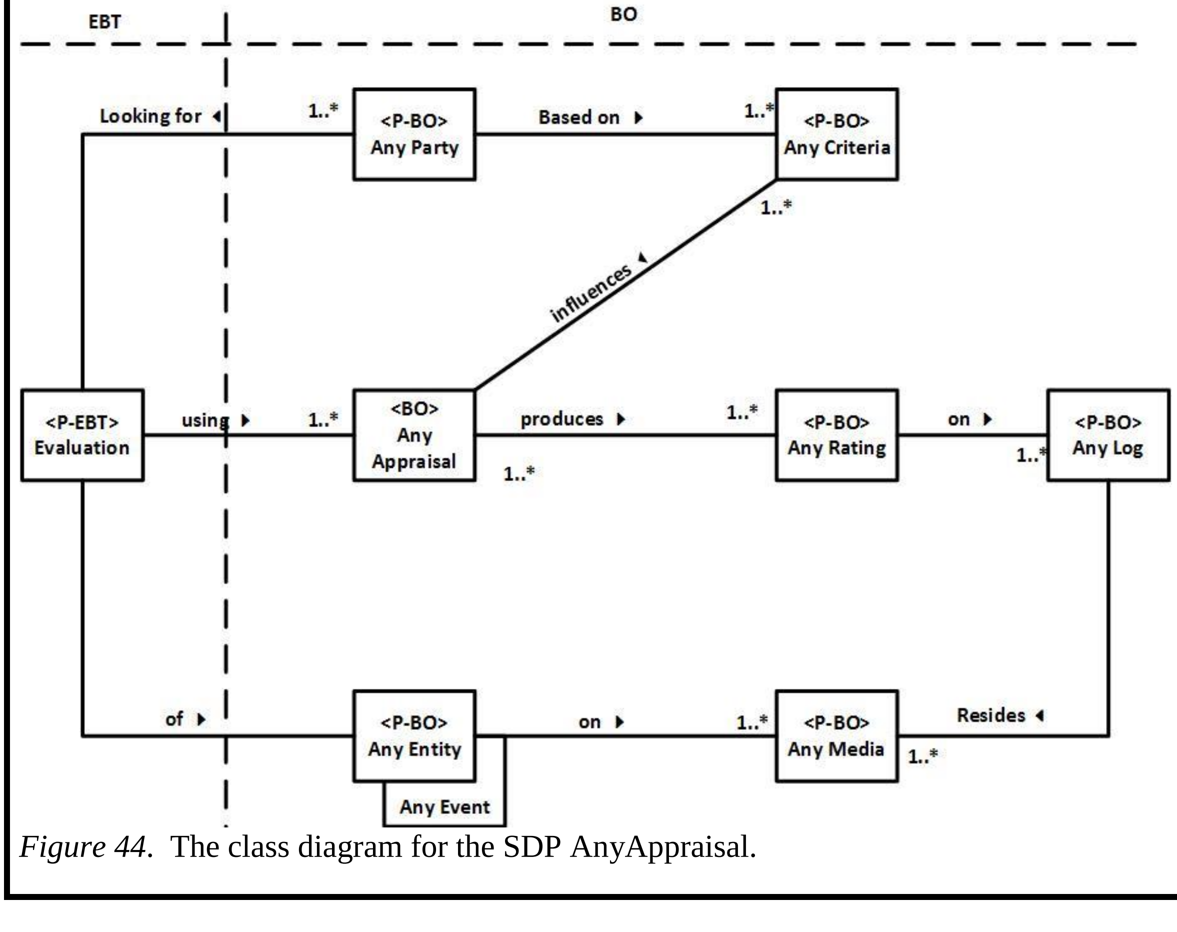


Figure 44. The class diagram for the SDP AnyAppraisal.

Class diagram description.

- AnyParty looks for an evaluation.
- The evaluation is done using AnyAppraisal of AnyEntity or AnyEvent.
- AnyAppraisal produces AnyRating.
- AnyEntity or AnyEvent is on AnyMedia.
- AnyMedia resides AnyLog.

Summary. This chapter identified and evaluated the most important components of AnyAppraisal that are general and common across many domains. AnyParty, AnyEntity, AnyCriteria, AnyMedia, AnyRating, AnyAppraisal and AnyMedia are the primary properties for Appraisal. These BOs are the workhorses that help developers achieve an appraisal. A developer should not create an appraisal application without these critical classes. This chapter provided a fundamental solution to the identified problem of developing a pattern that captures the idea of AnyAppraisal. The stable appraisal pattern may serve as a solid base for modeling any future pattern application.

Any Guideline SDP

Pattern Name: AnyGuideline SDP. Guideline is BO. It is an instruction or direction that explains how something needs to be carried out. A guideline is an aide, a sign, a law, an expression, a gesture, or a hint of a future strategy. Other names for guideline are direction, instruction, counseling, and navigation ("Guideline," 2016). The main reason for choosing this term is its comprehensiveness, as this term is appropriate for all the possible scenarios of any guideline. Figure 42 is a sample of an SDP that uses Guideline as its EBT.

Context. Guideline has applications in various domains such as finance, education, e-commerce, medicine, and programming. Some types of guidelines are medical guidelines, programming style guidelines, and writing guidelines. The following are some of the contexts where we can apply AnyGuideline SDP.

A medical guideline is a document that helps in making decisions regarding treatment (AnyAction) in the area of healthcare. A healthcare provider (AnyParty) follows the wound care algorithm (AnyGuideline) written by the National Guideline Clearinghouse (AnyParty) for the treatment of acute and chronic wounds (AnyEntity). The National Guideline Clearinghouse gives healthcare providers detailed information on clinical practice (Guidance). These guidelines make it easier for healthcare providers to prevent complications (AnyReason). The NGC also provides recommendations (AnyRule) that the guidelines support. Along with the recommendations, they list potential benefits and harms are listed.

There are also software development documents that contain a set of recommendations (Guidance) for software developers (AnyParty) to follow. Human Interface Guidelines benefit the developers by helping them make better applications (AnyEntity). The aim of these documents is to design (AnyAction) applications that are more intuitive to the client (AnyParty). The guidelines list the number of policies (AnyGuideline) that emerge human-computer interaction studies. The user experience can be improvised (AnyReason) with the help of these guidelines. For instance, Apple (AnyParty) has put forward human interface guidelines for Apple watch application designers (AnyParty) to understand how the Apple watch was designed and its specifications such as screen size and layout.

Functional requirements.

- Guidance represents the process of guiding. It can be a help or advice that tells us what to do. It has attributes such as directionProvided, function, and context. It has operations such as control(), teach(), navigate(), and counsel().
- AnyParty is considered the person or thing that provides or follows the guidelines such as an organization, a country, a political party, or a person. AnyParty has attributes such as name, location, catalog, and instructions. AnyParty has operations such as direct(), regulate(), and supervise().
- AnyActor represents anything that interacts with the AnyGuideline. It has attributes such as name, id, and role. It has operations such as followGuideline(), seekAdvice, and generateProtocol().
- AnyGuideline represents the instructions that tell AnyActor or AnyParty how something needs to be accomplished. It has attributes such as description, scope, and category. It has operations such as validateMethodology(), analyzeCost(), and recommend().
- AnyReason represents the explanation for why AnyGuideline exists. It has attributes such as evidence, benefits, harms, and description. It has operations such as qualify(), implement(), and identify().

- AnyRule represents the statements that let ANyParty or AnyActor know what is allowed and what is not. It has attributes such as standard and method. It has operations such as prevent(), control(), and regulate().
- AnyAction represents something done in regard to AnyGuideline. AnyAction has attributes such as outcome, style, level, and id. It has operations such as perform(), plan(), and operate().
- AnyEntity represents the object for which AnyGuideline is given. It has attributes such as id, name, and status. Its operations can be to update(), relationship(), and type().
- Guidelines can also be given to AnyEvent such as a football match. It has attributes such as name, category, and id. AnyEvent can have operations such as occur(), gather(), and involve()

Non-functional requirements.

- A guideline should be informative, which means it should provide some information about the product or service. For example, the guidelines developed by the World Health Organization dictate when children should receive immunizations. Other information about diseases is mentioned on their website.
- A guideline should be created in a timely manner. Guidance about something that no longer exists is pointless. For example, if a guideline is posted on how to use a payphone at the university, but the payphone has been removed and no longer exists, the guideline is not helpful or timely. The university needs to create guidelines that are more timely such as those regarding cell phone use on tests.

- A guideline needs justification with a good reason for its creation. In China, the one-child policy was implemented to control the population. The policy was enforced by imposing fines based on the income of the family if they had more than one-child. Critics consider the one-child policy a violation of human rights because humans have the right to determine the number of children in their family. The One-child Policy also forces individuals to undergo abortions, thereby making the policy unjustifiable.
- Guidelines should be appropriate, meaningful, and significant. While issuing AnyType of guidelines, relevant supplementary materials shall be included to support the guidelines. The guidelines should be relevant to the topic. For instance, when guidelines discuss programming issues, it is important to discuss programming styles. These are relevant because they help programmers avoid errors while coding. If the guidelines concern Java programming, it would not be relevant to discuss Python programming style.

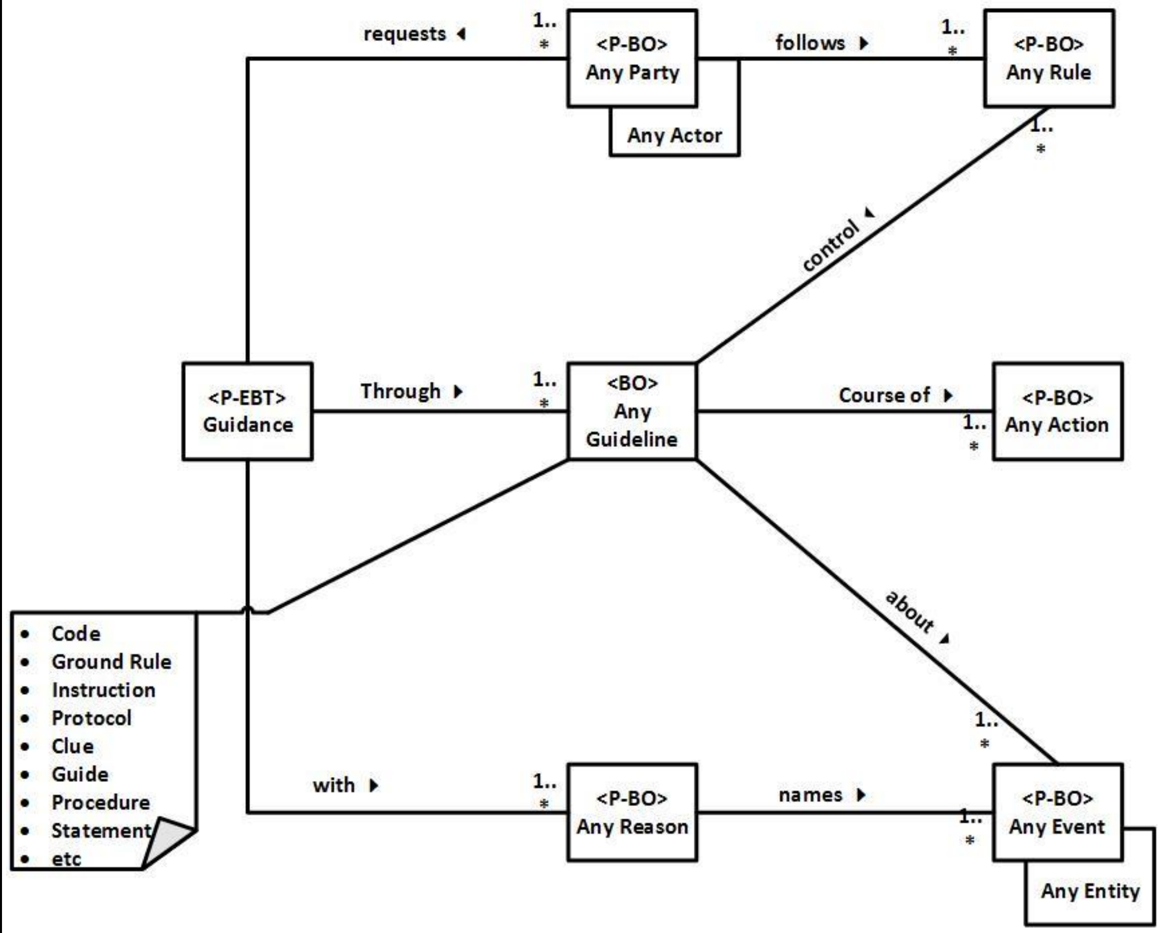


Figure 45. The diagram for the AnyGuideline SDP.

Class diagram description.

- AnyParty or AnyActor requests Guidance.
- Guidance is given through AnyGuideline.
- AnyReason exists for AnyGuideline.
- AnyGuideline is in charge of AnyAction.
- AnyGuideline is about AnyEntity or AnyEvent.
- AnyReason names AnyEntity or AnyEvent.

- AnyRule controls AnyGuideline.
- AnyParty follows AnyRule.

Summary. This thesis demonstrates how the core knowledge of AnyGuideline can be used in numerous applications. It demonstrates an SDP for AnyGuideline that is easy to maintain, cost effective, less time-consuming, robust, and flexible. This model can be extended to identical contexts, and is unique for its reusability, understandability, flexibility, and extensibility.

Any Model SDP: Pattern Documentation

Pattern name: AnyModel SDP. Model is a BO. AnyModel could be a standard for making comparisons or a replica to show how something would look. A model can be an example, a picture, a 3-D object, a duplicate, a pattern, an instance, a sample, and a representation. Other names for model are illustration, miniature, exemplar, copy, dummy, and imitation ("Model," 2016). The comprehensive nature of the term is the main reason for choosing it. This leads to a better SDP when model is its EBT.

Context. Model has applications in various domains such as mathematics, computer science, music, business, psychology, the arts, and entertainment. Some types of models are a scientific model, a 3-D model, a computer model, a system model, a conceptual model, a mathematical model, and role model. The following are some of the contexts where we can apply the AnyModel SDP.

A mathematical model is an illustration (AnyModel) of a system that represents how something works by using mathematical concepts. This process is known as mathematical modeling (Modeling). Mathematical equations tell about the behavior of future events. For example, a student (AnyParty) is given a math (AnyReason) problem as an assignment (AnyTest) by his professor (AnyParty) to find out the surface area of a cuboid without using a calculator (AnyCriteria). Solving such a problem would be a lot easier if the student breaks down the problem by drawing a sample figure (AnyView) of cuboid on a paper (AnyMedia) and then solving it.

A fashion model (AnyParty) is a person who promotes clothing and accessories (AnyEntity) as the latest fashion (AnyModel). These samples are made by fashion (AnyReason) designers (AnyParty) in order to display their work at a fashion show (AnyMedia). A fashion model may also model items in photographs (AnyView) for advertisement purposes. Celebrities (AnyParty) are common in the field of fashion modeling for promoting brands apart from their regular work. Supermodels (AnyType) are fashion models that are well known (AnyCriteria). It is easier for fashion experts and fashion buyers to analyze (AnyTest) designer clothes at a fashion event.

Functional requirements.

- Modeling refers to making representations of something. It has attributes such as name, make, and date. It has operations such as represent(), provideCopy(), and visualize().
- AnyActor represents someone or something involved in modeling. It has attributes such as name, id, and type. It has operations such as endorse(), design(), and illustrate().
- AnyParty represents the party involved. A party can be a country, political party, organization, or a person affiliated with an organization. AnyParty has attributes

such as name, id, and location. It has operations such as copy(), giveSample(), and imagine().

- AnyModel is the representation on a smaller scale of something. It has attributes such as type, version, and name. It has operations such as compare(), imitate(), and envision().
- AnyCriteria represents something that is behind a model. It has attributes such as description, name, and id. The operations of AnyCriteria are judge(), prove(), and measure().
- AnyTest represents the means by which one can measure the quality of something. It has attributes such as name, numberOfQuestions, and procedure. It has operations such as check(), examine(), and verify().
- AnyView represents a picture of something or even an opinion depending on the context. This class can have attributes such as context, mode, and range. It can have operations such as examine(), judge(), forSee(), and represent().
- AnyReason represents the instructions that tell AnyActor or AnyParty how something needs to be accomplished. It has attributes such as description, scope, and category. It has operations such as validateMethodology(), analyzeCost(), and recommend().
- AnyType represents the type of the model. AnyType has attributes such as id, name, interfaceList, methodList, and property. The operations are change(), categorize(), and subtype().

- AnyEntity represents the entity involved in AnyModel and is named by AnyType of AnyModel. The attributes can be id, name, and status. Its operations can be to update(), relationship(), and type().
- AnyEvent represents the event involved in AnyModel and is also named by AnyType of AnyModel. It has attributes such as name, occasion, type, and outcome. It has operations such as appear(), happen(), and arrange().
- AnyMedia represents the medium through which modeling takes place. AnyEntity or AnyEvent resides in AnyMedia. It has attributes such as name, type, status, and capability. It has operations such as broadcast(), capture(), display(), select(), navigate(), and remove().

Non-functional requirements.

- AnyModel should give assistance. Consider a situation where a civil engineer draws the model of a house to show the clients how the house would be constructed. The model would visually represent the house and help the client to understand the layout.
- A model should acceptable to the client. Consider a supermodel who has lots of experience modeling. But, if a particular set of designer clothes do not fit the shape or style of the supermodel, then it would be wise to choose a model on whom the clothes fit well. At the same time, if an architectect designs an office building and his child sits on it and breaks it, it would not be acceptable to display that model for his company.

- AnyModel should be relevant. A 2-D model is a great way to represent mathematical models when solving problems related to 2-D geometry, but when representations of 3-D models are necessary such as for online gaming, a 2-D model would be irrelevant.
- AnyModel should be complete in all respects. AnyModel that is incomplete may not make any sense. A person finds it to be very worthless. If a computer programmer is working on a model game demo and only writes half the code, it does not make sense because it would be missing many key components.
- AnyModel should be consistent in its mode of working. For example, a sculptor who wants to create a model of a tree should not change mediums and work with a material he or she has never worked with before. If the person who commissions the sculpture is expecting a granite sculpture of a lifelike tree, and the sculptor gives the buyer a plaster tree or one made with unfinished granite, it will cause problems. Also, if the model is designed in an abstract way, it will not be consistent with the realistic product desired.

Solution.

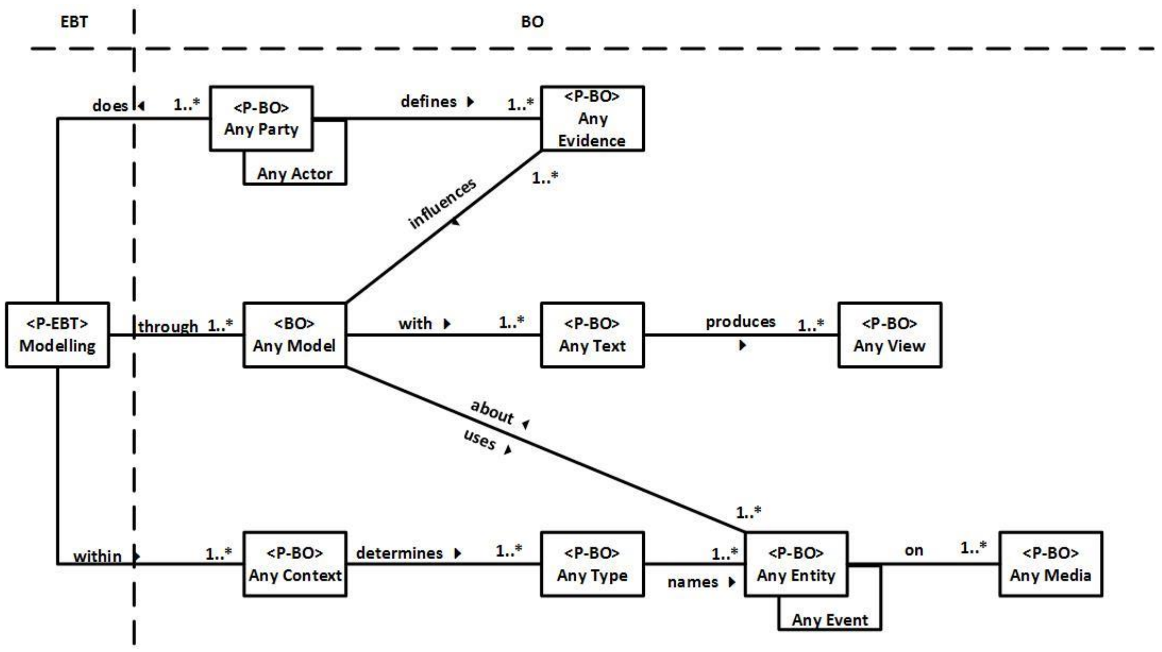


Figure 46. The diagram for the AnyModel SDP.

Class diagram description.

- AnyParty or AnyActor does modeling.
- Modeling is done through AnyModel and within AnyReason.
- AnyModel uses AnyEntity or AnyEvent with AnyTest.
- AnyEvidence influences AnyModel.
- AnyReason determines AnyType.
- AnyType names AnyEntity or AnyEvent.
- AnyEntity or AnyEvent is about AnyModel on AnyMedia.
- AnyTest produces AnyView.

Summary. The mid-sized template generated in this section exhibits how the core knowledge of AnyModel can be used for numerous applications. With proper identification and detection, it is possible to cull out the main theme for the pattern along with its associated BOs and IOs. This also interprets the SDP for AnyModel in an effective manner. This model can also be extended to identical contexts. Stability, reusability, and robustness make this contribution valuable.

Any Review SDP: Pattern Documentation

Pattern name: AnyReview SDP. Review is BO. A review is examining the quality of something. It is a report that presents opinions about a product, service, or anything else. Other names for review are analysis, audit, check, inspection, and report ("Review," 2016). The reason for choosing the term *review* is because it is a more generic term than others.

Context. Review has applications in various domains such as entertainment, magazines, software, and electronic devices. Some types of reviews are book reviews, music reviews, performance reviews, recording reviews, composition reviews, movie reviews, and concert reviews. The following are some of the contexts where we can apply the AnyReview SDP,

Consider a situation where a cell phone reviewer (AnyParty) evaluates the Samsung Galaxy S6 (AnyEntity) and publishes the review on a website (AnyMedia). In order to create a review on Samsung phone, a comparison (Evaluation) of similar products occurs. Analysis (AnyReview) of the phone's features such as its battery life, its physical appearance, its camera, and its design are performed. Key points (AnyRating) are given for each of the features. Such a comprehensive analysis would benefit (AnyReason) the buyers (AnyParty) who plan to buy the phone. While posting an online

(AnyType) review on a website (AnyMedia) for viewers, the post does not contain any hate message or plagiarism (AnyRule). The content of the website is stored in a database (AnyLog) on the Cloud so that one can search for it later whenever necessary.

Another example is the software review (AnyReview), which is a process where the manager (AnyParty) and other interested representatives examine a software product (AnyEntity) for approval (AnyReason). A formal (AnyType) software review has a set of evaluations including preparation, examination, and follow-up. A software product could be a document, source code, or a standard. If the project design needs review, then the manager would do an inspection (AnyReview) and suggest any changes.

Software peer review is a methodology of software review, where colleagues evaluate the work of another colleague (AnyParty). Peers are asked to rank (AnyRating) the software based on their assessment of the quality of the software. The value of any software review is that the defects within the software are discovered by following the review procedures (AnyRule). The reviews can be collected through online review tools such as Survey Monkey (AnyMedia) and the reviews collected through this tool would present a custom record (AnyLog) available to the manager to view.

Functional requirements.

- Evaluation determines the importance or worth of something. Evaluation is done for AnyReason and has attributes such as worth, name, opinion, and decision. It has operations such as determine(), setValue(), and judge().
- AnyParty does evaluations based on AnyRule. It represents a country, a political party, a government, or an individual belonging to an organization. AnyParty

generally has a name, type, and qualityOfAssessment. It has operations such as expressOpinion(), criticize(), and comment().

- AnyActor also does evaluations based on AnyRule. AnyActor represents someone or something that is involved in AnyReview. AnyActor has attributes such as name, conetxtOfEvaluation, experience, and level. It has operations such as findFault(), appreciateWork(), and analyze().
- AnyRule represents the standards for AnyReview. AnyRule that is meant for AnyReview is commonly known as a business rules. It has attributes such as duration, description, regulation, and authority. AnyRule has operations such as govern(), administerGuideline(), and prohibit().
- AnyReview represents a critical look at AnyEntity or AnyEvent. AnyReview has attributes such as context, nature, quality, and isUseful. It has operations such as check(), provideValuation(), and giveFeedback().
- AnyReason represents the explanation for the evaluation. It has attributes such as evidence, benefits, harms, and description. It has operations such as qualify(), implement(), and identify().
- AnyRating represents a classification indicated by its capability. It has attributes such as position, category, and context. It has operations such as classify(), compare(), and assess().
- AnyType represents the variety of types AnyReview can have and is determined by AnyReason. AnyType has attributes such as id, name, interfaceList,

methodList, and property. The operations are change(), categorize(), and subtype().

- AnyLog represents the number of AnyReview entries. AnyRating is stored on AnyLog. It has attributes such as name, type, criteria, numberOfEntries, size, and location. It has operations such as edit(), openLog, search(), close(), remove(), and insert().
- AnyMedia represents the medium through which AnyReview takes place. AnyEntity or AnyEvent resides on AnyMedia. It has attributes such as name, type, status, and capability. It has operations such as broadcast(), capture(), display(), select(), navigate(), and remove().
- AnyReview is done for AnyEntity and is named by AnyType of AnyReview. It has attributes such as id, name, and status. Its operations can be to update(), relationship(), and type().
- AnyReview is also done for AnyEvent and is named by AnyType of review similar to AnyEntity. It has attributes such as name, occasion, type, and outcome. It has operations such as appear(), happen(), and arrange().

Non-functional requirements.

- A review should be informative and provide useful information about the product or service being reviewed. If someone writes a review about a product, the review should provide information about the product and reasons supporting their opinions. If something about a product does not work so well it should be mentioned in the review, so that other purchasers can be aware of the issue.

- A review should be relevant. Consider a situation where a film critic is supposed to review Star Wars, but watches X-men instead. If the film critic writes a review for Star Wars, it will not be relevant because the critic watched the wrong movie.
- A review should be accessible. If one writes a review about a movie for a magazine but the magazine never gets published, then no one will read the review. A review needs to be accessible to AnyParty or AnyActor.
- A review should solve a problem by providing help. The number of customer reviews has increased dramatically in the past few years. The reviews are expected to be given by experts, or at least people who have used the product or service. They should be helpful to the consumer when he or she is making purchase decisions. For example, reviews on Amazon are ranked based on how helpful they are to other customers. AnyParty or AnyActor that gives reviews can be ranked based on how helpful their reviews are to other buyers.
- A review should be given on time. A review about an outdated item is of no use. Consider a situation, where a student asks a professor to review his project before he or she submits it to a competition. The student can improve the project based on the professor's reviews if they are received in a timely manner. There is no use in receiving the review from the professor after the submission date. Therefore, timeliness is important for any review.

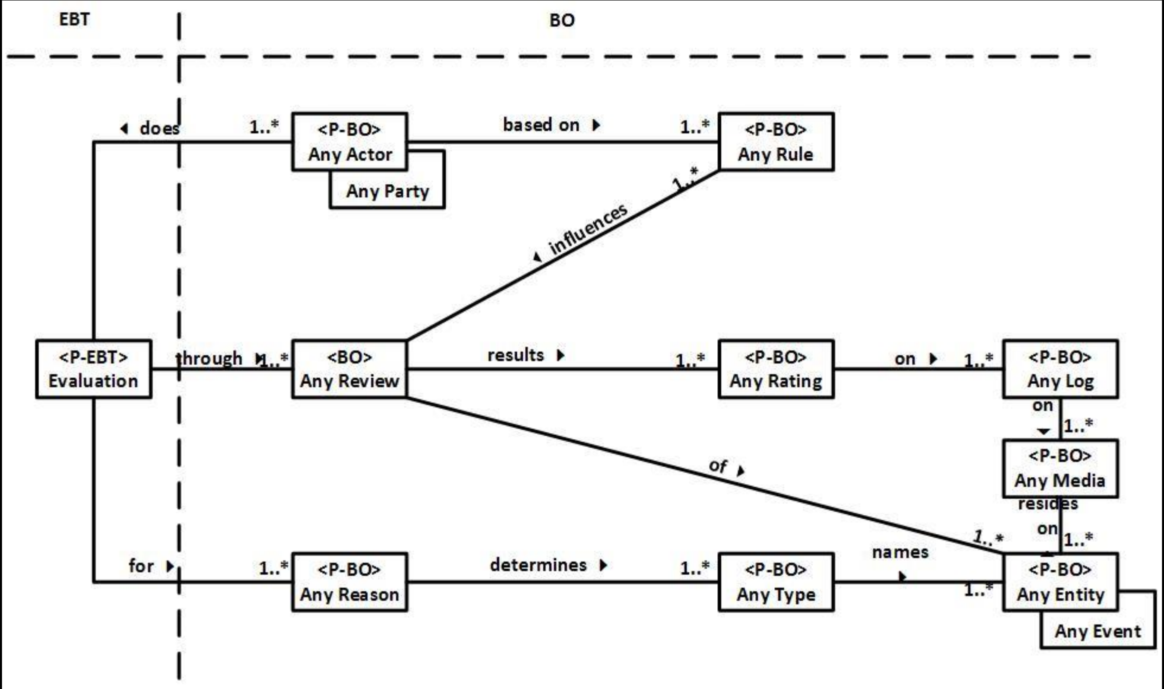


Figure 47. The diagram of the SDP for AnyReview.

Class diagram description:

- Evaluation is done by AnyParty or AnyActor through AnyReview for AnyReason.
- AnyParty or AnyActor is based on AnyRule.
- AnyReview results in AnyRating and is about AnyEntity or AnyEvent.
- AnyReason determines AnyType.
- AnyRule influences AnyReview.
- AnyType names AnyEntity or AnyEvent.
- AnyReview resides on AnyMedia.
- AnyLog records AnyMedia.

Summary. In the context of creating a pattern template, the most critical need is to identify the most appropriate theme for review, and it is possible to extract it only through an in-depth evaluation of different review terms. An intensive examination of review has helped us in identifying the main theme for this chapter, which is evaluation. The mid-size template provided in this chapter exhibits how core knowledge of AnyReview can be used for numerous applications. It interprets SDP for AnyReview in an effective manner. This model can also be extended to identical contexts. Reusability of the pattern itself is a very big contribution apart from savings in time, effort and money.

Any View SDP: Pattern Documentation

Pattern name: AnyView SDP. View is a BO. A view is an opinion about something. A view can be something that is seen, believed, outlined, or just spectacle. Other names for view are aspect, glimpse, outlook, perspective or picture ("View," 2016). Generality is the main reason for choosing this term, as this term is appropriate for all the possible scenarios of AnyView. This lead to an SDP using viewing as its EBT.

Context. View has numerous applications in various domains, such as entertainment, magazines, and software engineering. Some types of views are a graphical view, an SQL view, a model-view-controller design pattern, and a point of view,. The following are some of the contexts where we can apply Any View SDP.

Computer view, commonly known as computer vision, is a process where a computer (AnyActor) can recognize images like human beings. It deals not only with recognizing (AnyView), but also with processing, analyzing, and understanding

(Viewing) the images to make decisions. This type of object recognition techniques can be of great use in many different areas. One example would be that it could help a blind person (AnyActor) in recognizing high dimensional objects without touching (AnyCriteria) them. A common way of recognizing the objects for blind people is by touching and feeling the objects, but computer view or computer vision eliminates the necessity of touching every object because it can recognize objects just by processing images. A newspaper can be read aloud by image processing technology, which is a subset of computer view, just by putting the newspaper in front of the camera. Similar technology is also being used to produce unmanned vehicles that run by processing the view. The view is captured by them helping the vehicle to run without a human. A computer captures the images (AnyEntity) by using a camera (AnyMedia). The camera has different modes of operation such as automatic (AnyMode), manual, or night mode.

Bird’s-eye view (AnyView) is a view from a high altitude, as viewed by a bird when it is flying. The observer is regarded as a flying animal in this case. Bird’s-eye view has various modes such as a photograph, video (AnyMode) or a drawing. A view from the top of a high mountain or a tower is also considered a bird’s-eye view. One uses this kind of view when preparing a map, layout, film, or a blueprint. In filmmaking, the cameraperson (AnyParty) uses a bird’s-eye shot, say, to capture a battle scene, (AnyEvent) to view the actors (AnyParty) from above, and to be able to move near or away from the subjects. Generally, for such shots, the camera needs to be taken to a higher altitude (AnyCriteria) than the usual. For taking cameras to a high altitude, the director and cameraperson mostly use a crane (AnyMedia) to achieve specific shots.

Functional requirements.

- Viewing represents the inspection of something. Viewing has an elementOfInspection, type, and description. It can have operations such as survey(), inspect(), explore(), and observe().
- AnyView represents an instance of vision. AnyView can be a picture of something or even an opinion depending on the context. The AnyView class can have attributes such as context, mode, and range. It can have operations such as examine(), judge(), foresee(), and represent().
- AnyParty represents a country, political party, government, and persons belonging to an organization. AnyParty has a name, location, and phoneNumber. It have operations such as monitor(), analyze(), and check().
- AnyActor participates in various activities. AnyActor has a name, id, and category. It has operations such as interact(), look(), plan(), and conclude().
- AnyCriteria represents a class that is used to make decisions. It has attributes such as standard, principle, proof, and judgment. It has operations such as test(), confirm(), verify(), and assess().
- AnyMedia represents the media though which AnyView takes place. It has operations attributes such as name, availability, and isAvailable. It has operations such as display(), store(), capture(), broadcast(), and connect().
- AnyEntity represents the entity that is viewed. An entity can be any visible object. It has attributes such as name, type, and position. It has operations such as status(), performFunction(), and update().

- AnyEvent represents any viewable occurrence. An event is something that happens. It has attributes such as name, occasion, type, and outcome. It has operations such as appear(), happen(), and arrange()
- AnyMode class represents the manner of doing something. It has attributes such as method, type, condition, style, and name. It has operations such as provideChoice(), governQuality(), and regulateFashion().

Non-functional requirements.

- A view should be high-quality. It should have all the essential characteristics. Because of the increasing demand in consumer electronics, 3-D display systems have come into existence. According to the high-quality viewing synthesis algorithm (Lai, Lai, & Lyn, 2012, p. 5), frames with virtual views can be corrected. The algorithm for high-quality view synthesis helps in capturing good human view perception.
- A view should accomplish a purpose and be effective. It should give the expected result. For example, in the case of a human-computer interaction which uses a computer view or computer vision technology, effectiveness would mean that a system should be able to complete the tasks given to it in an accurate manner. In software engineering, when we are building system architecture, the view model needs to be effective or else the framework lacks coherence.
- A view should be easily obtainable. We should be able to find it easily. If there is an automated vehicle which can view, process, and analyze its surrounding in order to drive but this vehicle is inaccessible to the majority of car owners, it is

not helpful. Also, the computer view should be accessible by the unmanned vehicle in order to function. Finally, a cameraperson cannot take a point-of-view shot, when it is not accessible to him.

The following class diagram provides details about views, AnyView, and Viewing.

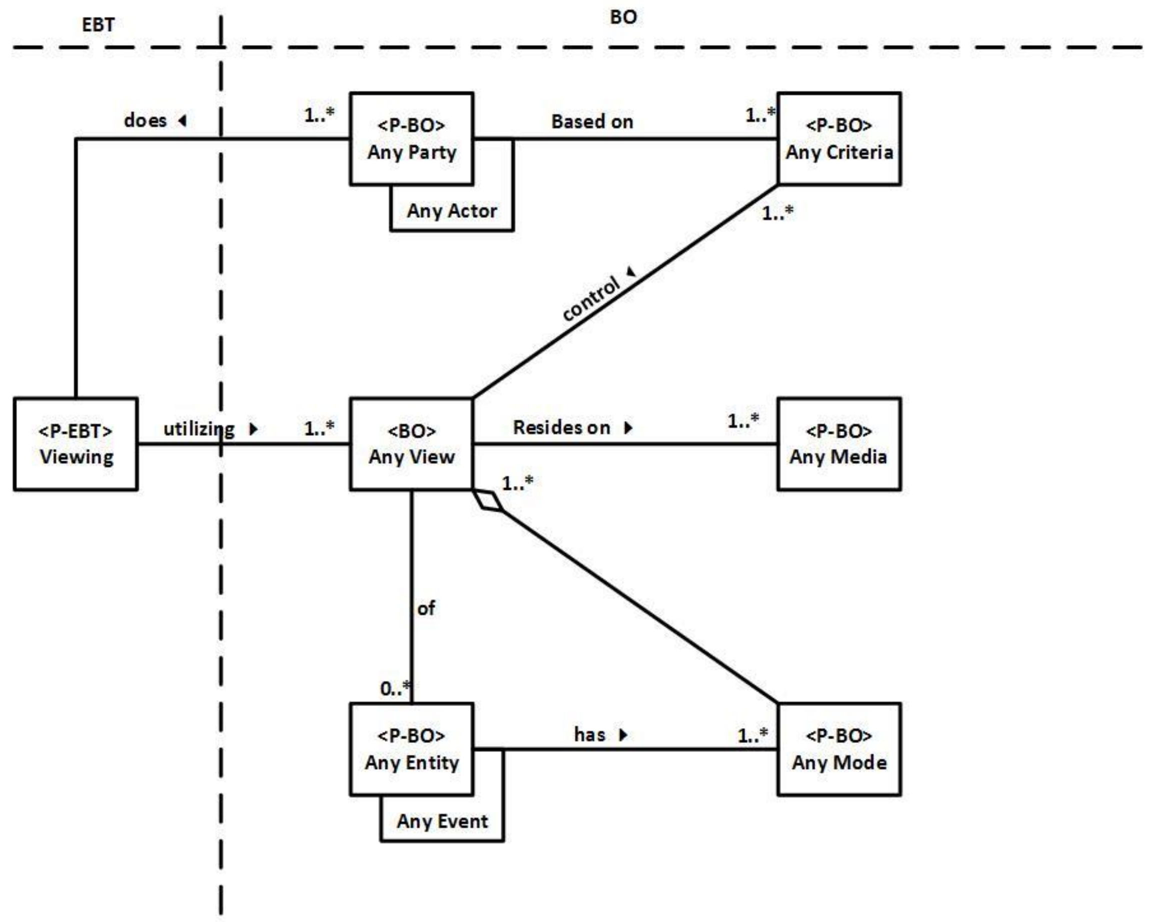


Figure 48. The class diagram for the AnyView SDP, showing the EBT and the BOs.

Class diagram description.

- AnyParty or AnyActor views something based on AnyCriteria.
- Viewing utilizes AnyView.

- AnyView resides on AnyMedia, and AnyCriteria controls it.
- AnyView is of AnyEntity or AnyEvent.
- AnyEntity or AnyEvent has AnyMode.

Summary. The mid-size template demonstrates how one can identify the core knowledge of AnyView and in what manner it can be used to create applications for many different scenarios. This model can be extended to identical contexts due to its stability, reusability, robustness, reduced time for creation, lower cost, and reduced effort to create required patterns.

Any Violation SDP

Pattern name: AnyViolation SDP. Violation is a BO. A violation is an action of doing something that is not allowed or is illegal. A violation could be an act of doing something unconstitutional, criminal, irregular, illicit, prohibited and so on. Other names for violation are abuse, contravention, infringement and offense ("Violation," 2016). Generality is the main reason for choosing this term, as this term is appropriate for all the possible scenarios of violation. This lead to an SDP using Infringement as its EBT

Context. Violation has numerous applications in various domains such as law, sciences, human rights, and telecommunications and so on. Some types of violations are a parking violation, an infraction, and a rape. The following are some of the contexts, where we can apply Any Violation SDP.

Consider a situation where a motor vehicle (AnyEntity) is parked in a restricted place such as in the middle of a highway. There are restrictions (AnyRule) on where and where not to park and to break them is regarded as an offense. a police officer

(AnyParty) usually cites the offender (AnyParty). A parking ticket is issued to the offender, who violated the California (AnyState) traffic laws. Parking rules are posted on roads in the form of signs (AnyMedia) such as ‘No Parking.’ Breaking such rules (AnyRule) can lead to a heavy (AnyType) fine.

Consider the situation in North Korea (AnyParty). Kim Jong-un conducted mass atrocities (AnyEvent), such as forcing women to drown their own babies in a bucket of water. This type of human rights abuse (AnyViolation) on a large scale (AnyType) was seriously condemned by the United Nations. This is the violation of moral principles stated by the United Nations Organization (AnyParty). These principles are protected by international law (AnyRule). These are the fundamental rights that every person has regardless of his or her location (AnyState), religion, race, and status.

Functional requirements.

- Infringement represents the act of breaking the law. Infringement has attributes such as lawViolated, punishment, and offenderName. It has operations such as intrude(), rob(), and trespass().
- AnyParty represents a party that is involved in the violation. A party can be a country, political party, organization, or a person affiliated with an organization. AnyParty has attributes such as name, id, and location. It has operations such as commitCrime(), breakLaw(), and disobeyRule().
- AnyViolation represents the act of violating something. AnyViolation has attributes such as nameOfAbuse, violationType, and crime. It has operations such as offend(), abuse(), break(), and condemn().

- AnyType represents the type of infringement. AnyType has attributes such as id, name, interfaceList, methodList, and property. The operations are change(), categorize(), and subtype().
- AnyRule represents the standards for any violation. It has attributes such as duration, description, regulation, and authority. AnyRule has operations such as govern(), administerLaw(), and prohibit().
- AnyState represents the state in which violation takes place. AnyState has attributes such as name, location, size, and population. It has operations such as serveNation(), declare(), and maintain().
- AnyEntity represents the entity that is involved in any violation and is named by the type of violation. The attributes can be id, name, and status. Its operations can be to update(), relationship(), and type().
- AnyEvent represents the event that is involved in any violation and is also named by the type of violation. An event can be a soccer match being played between two countries. It has attributes such as name, occasion, type, and outcome. It has operations such as appear(), happen(), and arrange().

Non-functional requirements.

- A violation should be preventable. One should be able to stop a violation from happening. It is not so easy to prevent every violation. Consider a violation such as driving under the influence, which is a driving offense. Suggesting an alternative method of reaching the destination, such as a cab, can prevent drunk driving. Another way of preventing someone from driving drunk is to send him or her to his or her destination, via public transportation with a sober friend. Thus, any violation has ways of prevention.

- A violation shall be such that it could be discovered or noticed when it happens. Copying others work and showing it as your work is a copyright infringement and can be detected in software such as Turnitin, which detects any possible plagiarism.
- We should be able to control violence, which means we should be able to direct the behavior of AnyParty or AnyActor the way we want to. The drivers should obey the traffic control devices. An example of violation of a traffic control device is running a red light. If a driver does not obey the traffic signals, he or she be charged with a violation of the control device. Therefore, every violation can be controlled.

Solution.

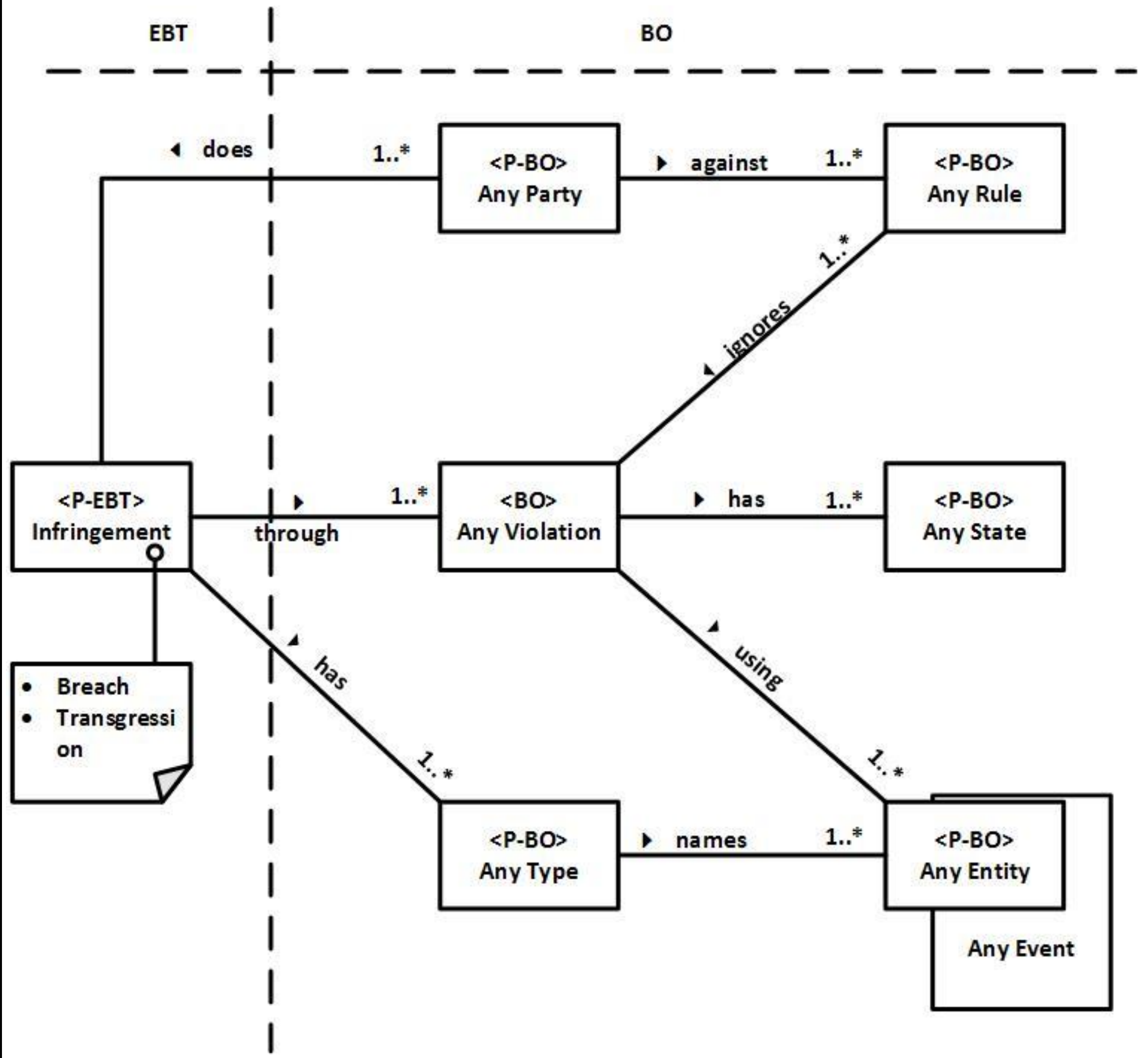


Figure 49. The class diagram of AnyViolation SDP, displaying the EBTs and BOs.

Class diagram description.

- AnyParty infringes against AnyRule.
- Infringement occurs through AnyViolation and AnyType.
- AnyType names AnyEntity or AnyEvent.
- AnyViolation has AnyState using AnyEntity or AnyEvent.

- AnyRule ignores AnyViolation.

Summary. The template in this chapter exhibits how the core knowledge of any violation can be used for numerous applications under different contexts. The pattern template provides a brief view of how a developer can extract the EBT for the system and link it with other essential BOs and IOs. The resulting pattern can be used in many other contexts where the AnyViolation concept is used. It interprets the SDP for AnyViolation in an effective manner.

The Fulfillment SAP

Pattern name. A fulfillment can be a feeling of accomplishment, completion, and ability. Other names for fulfillment are achievement, contentment, and gratification ("Gratification," 2016). Generality is the main reason for choosing this term, as this term is appropriate for all the possible scenarios of Fulfillment that leads us to Fulfillment SAP.

Context. Fulfillment has applications in various domains such as e-commerce, real estate, and customer satisfaction. Some types of fulfillment are service fulfillment and many more. The following are some of the contexts where we can apply fulfillment as SAP.

- Order Fulfillment: Order fulfillment (Fulfillment) is a common process from point of sale inquiry until the delivery of the product. It is described as a narrow act of distribution (AnyMechanism) where a purchaser makes a general inquiry about the product to the actual transfer of the shipment to the purchaser. Consider a scenario where a purchaser (AnyParty) would try to buy something on the

Internet. Such a purchase would be an online (AnyType) order. The product being purchased from such online orders could be a laptop (AnyEntity). Online orders are made through different websites. One website is Amazon (AnyParty), which is famous in the field of e-commerce. Once an order is placed on Amazon, the product is shipped to the purchaser. There are different modes of shipping depending on the number of days (AnyCriteria) it takes to ship. Once the laptop has been shipped, then the order is considered to be fulfilled. Also the desire of the purchaser to have a new laptop and play (AnyWant) games on it would be fulfilled.

- Contract Fulfillment: Contract fulfillment (Fulfillment) means to fulfill the terms (AnyWant) of a contract. It can also be defined as adhering to the conditions (AnyCriteria) of your agreement for the entire term. For instance, if an employer (AnyParty) employs contractors (AnyParty), they would expect the job to be done in the right time frame. If that does not happen, then the employee could be refused from being paid. When signing (AnyMechanism) a contract (AnyEntity) between a contractor and an employer (AnyParty) takes place we call it a contractual (AnyType) obligation (AnyCriteria). The contract would have statements as in what the company requires the employee to do. The contract which has to be signed between both of the parties would consist of different articles, which describe the terms and conditions, services to be performed, compensation, terminations terms, confidentiality, general provisions, ownership, and so on.

Functional requirements.

- Fulfillment represents a sensation of satisfaction. It is a feeling that arises when a desire is achieved. Fulfillment has a condition, type, description, id, and restriction. A fulfillment allows() a party to achieve() its goal and complete() something. A fulfillment can also mean to obeyCommand() or to performDuty().
- AnyParty is a legal user like an organization, a person, a government, or a political party. AnyParty has a name, contact information, and phoneNumber. An employee of an organization can participate() in the company projects, setCriteria() on his team by using collectData(), and interact() with other employees. A party usually can completePromise(), executeRequest(), and satisfyDemands() of something.
- AnyCriteria represents something that is used for settling a judgment or choice. Any Criteria has attributes such as description, name, and id. The operations of criteria can be judge(), prove(), and measure(). AnyCriteria can evaluate() and test() something.
- Fulfillment is usually triggered by AnyMechanism and is a means of generating required results. AnyMechanism has a context, id, name, status, and application. It has operations such as activate(), attach(), detach(), and execute().
- Fulfillment can take variety of types that make use of AnyEntity and AnyEvent. There are different types of fulfillment because of various aspects such as wants and criteria. Type has attributes such as id, name, interfaceList, methodList and property. The operations are change(), classify() and subtype().

- Fulfillment can be achieved by AnyEntity such as a product from an online store. Also fulfillment can be achieved through events such as marriage. The attributes can be id, name, and status. Its operations can be to update(), relationship(), and type().
- Fulfillment can happen for a AnyEvent such as a sports competition. AnyEvent has a name, type, and id. An event can have operations such as occur(), gather(), and involve().
- AnyWant represents a desire that AnyParty has and needs to be fulfilled for satisfaction. The attributes are id, description, and type. AnyWant has operations such as desireToPossess(), need(), crave(), choose(), and wishFor().

Non-functional requirements.

- Any fulfillment should be on time and shall happen at the right time. For instance, the timeliness rule in bookkeeping alludes to the requirement for bookkeeping data to be introduced to the clients within enough time to fulfill their desires. Timeliness of bookkeeping data is very important since data that is introduced in a timely manner is more pertinent to clients. Postponement of data procurement has a tendency to render it less applicable to the clients.
- A fulfillment should be worthy of being accepted. It should be fairly good and satisfactory. For instance, in contract fulfillment, some conditions should be agreed upon by both parties of the contract. Say there is a contract employee who is hired for a specific purpose and paid only if that purpose is met. Then, the

employee should accept the task. He accepts that he is able to complete the given task. Without his acceptance, he would not be hired.

- Fulfillment should be practical. It has to be possible, conceivable, and workable. For example, an online order can be shipped to a remote place only if postal service exists at that location. If we try to send a shipment to a different country using a local courier service, then the shipment cannot be sent. When we need to fulfill an order, we have to choose a service that is doable.
- A fulfillment needs to be obtainable. When one dreams of becoming something, it is important to fulfill that dream. It is possible only when the dream is achievable. One cannot simply dream of something impossible, such as flying without a device, and then work toward fulfilling it even after knowing that it is not achievable. In a software company, when a team of engineers is asked to make software, the team is asked how much time it should take.

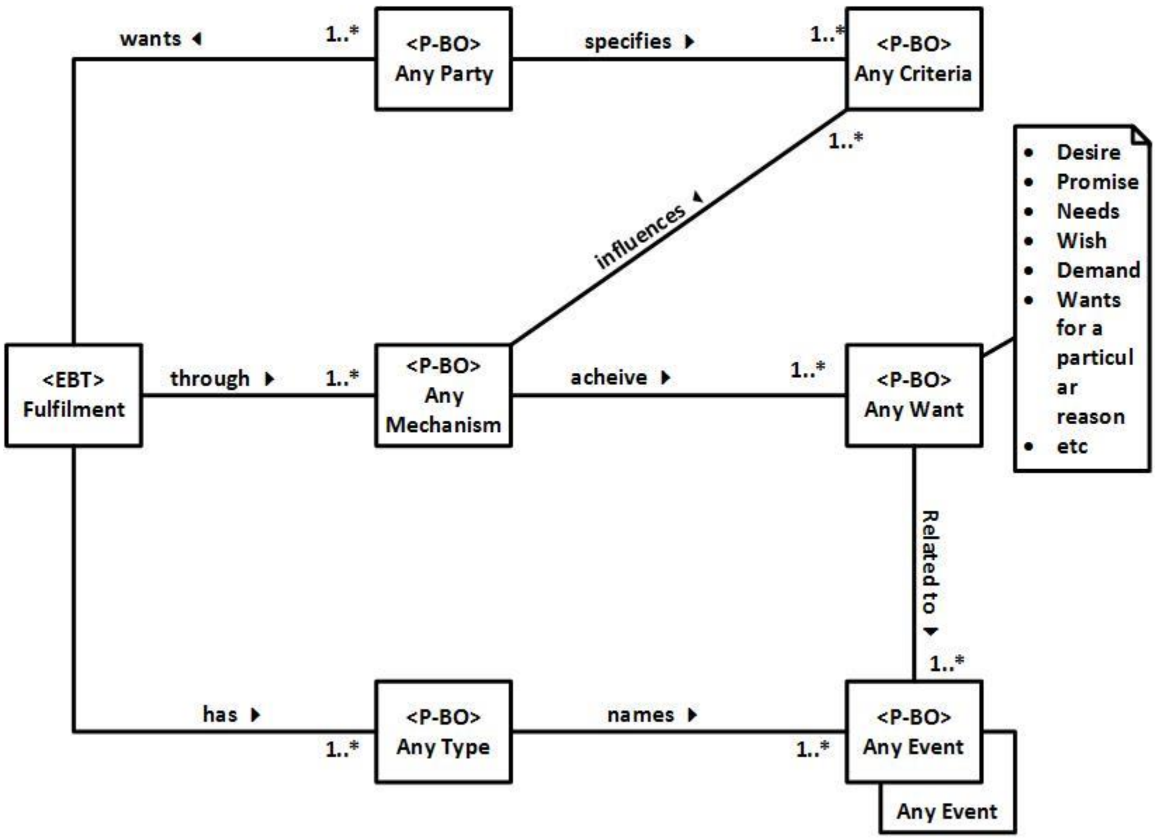


Figure 50. A class diagram of the fulfillment SAP.

Class diagram description.

- AnyParty wants fulfillment and specifies AnyCriteria.
- AnyCriteria influences AnyMechanism.
- Fulfillment has AnyType and is achieved through AnyMechanism.
- AnyMechanism achieves AnyWant.
- AnyWant relates to AnyEntity or AnyEvent.
- AnyEvent or AnyEntity is named AnyType

Summary. The fulfillment stable pattern is not only stable and robust but also truly reusable and flexible. The core concepts of this pattern are evaluated by using the basic principles of SAPs, while innumerable BOs help developers in modeling core concepts of fulfillment, including AnyParty, AnyEntity, and AnyMechanism. Once these BOs are extracted, it was easy to develop a stable pattern for fulfillment that could be applied to any domain and in any context. Compared to a traditional pattern approach, the pattern developed here lends a clear advantage of reusability, extendibility. and scalability. This pattern model is more sophisticated and mature when compared to the traditional pattern. Another advantage is the fulfillment pattern's ability to hook up to numerouse applications where the core concepts are applicable and relevant. In the patterns developed here, the importance of EBTs and BOs is obvious enough to suggest that they are generic enough to be used in many other domains. While finding out an EBT and related BOs are challenging exercises, plugging necessary IOs to any application developed through the fulfillment stable patterns itself is another dicey exercise. With patterns highlighted here, it should be possible to use any application repeatedly and without any major changes just by hooking important IOs. This chapter provides two different applications by applying the principles of fulfillment stable patterns without introducing any major changes or alterations; they, in fact, use their self-generated IOs that can be reused without introducing any major changes.

Promotion SAP: Pattern Documentation

Pattern name: Promotion SAP. Promotion is an EBT. It is defined as creating awareness about something in particular. A promotion can be creating alertness, conscious, information and so on. Other names for promotion are advertising, popularizing, publicizing, and endorsing ("Promotion," 2016). Generality is the main reason for choosing this term, as this term is appropriate for all the possible scenarios of promotion. This lead to an SAP.

Context. Promotion has applications in various domains such as marketing, entertainment, and film. Some types of promotion are print media, electronic media, and verbal promotion. The following are some of the contexts where we can apply promotion as SAP.

Whenever a new film (AnyEntity) is made, it needs to be promoted to let the moviegoers (AnyParty) know that such a film exists. A film can be advertised (Promotion) in theaters, on television, on the radio, and in print media. Film distributors (AnyParty) are responsible for promoting the film because they distribute promotional material to cinemas and theaters. There are different trends in promoting a film, such as releasing trailer highlights (AnyMechanism) on social media (AnyType) platforms. The duration of a trailer is less than three minutes (AnyRule), and its creation is a common practice followed by all film promoters (AnyParty). The Oscar (AnyAward) is a popular American award ceremony for honoring (AnyReason) achievements in the film industry.

Product promotion (Promotion) increases the demand for a product and allows consumers (AnyParty) to notice the product. Consider a new television (AnyEntity) by Samsung (AnyParty) that needs to be promoted. It can be done in print (AnyType) media. An advertisement (Promotion) for the television would be published (AnyMechanism) in newspapers and magazines to raise customer awareness (AnyReason). Even product promotion has rules and regulations to be followed. In the

United States, several advertisements have severe restrictions (AnyRule).For example, an alcohol advertisement that promotes excessive drinking is forbidden.

Functional requirements.

- Promotion represents the advancement in position. Promotion has attributes such as name, context, and position. It has operations such as advertise(), encourage(), and publicize().
- AnyParty represents a party that is involved in promotion. A party can be a country, political party, organization, or a person affiliated with an organization. AnyParty has attributes such as name, id, and location. It has operations such as moveHigher(), advance(), and sponsor().
- AnyActor represents someone or something that is involved in a promotion. It has attributes such as name, id, and type. It has operations such as endorse(), upgrade(), and elevate().
- AnyMechanism represents the method used for promotion. It has attributes such as name, context, description, application, and status. It has operations such as execute(), attach(), detach(), and activate().
- Every promotion has AnyReason for it, and this class represents the reason for selling. It has attributes such as description, proof, and justification. It has operations such as conclude(), examine(), and resolve().
- AnyRule represents the standards for any promotion. It has attributes such as duration, description, regulation, and authority. AnyRule has operations such as govern(), administerGuideline(), and prohibit().

- AnyCriteria defines a standard based. It has attributes such as description, name, and id. The operations of criteria can be judge(), prove(), and measure().
- AnyAward represents something that is awarded as a result of AnyMechanism. It has attributes such as name, and category.
- AnyType represents the various types of promotion and is determined by AnyReason. AnyType has attributes such as id, name, interfaceList, methodList, and property. The operations are change(), categorize(), and subtype().
- The promotion is done for AnyEntity and is named by AnyType of promotion. It has attributes such as id, name, and status. Its operations can be update(), relationship(), and type().
- The promotion is done for AnyEvent and is named by AnyType of promotion. It has attributes such as name, occasion, type, and outcome. Its operations can be to appear(), happen(), and arrange().

Non-functional requirements.

- The promotion should be accepted. Reposting a movie trailer for promotional purposes is a violation of copyright. Such a promotion is not acceptable. Email spam is another type of unacceptable promotion where junk emails are sent to numerous recipients in the form of a promotion.
- The promotion has to be relevant. It should be appropriate, meaningful, and significant. Consider a scenario where an American bank is promoting itself in another country where it has no branches. A promotion should be relevant to have value.

- The promotion should be effective. It should show results that are necessary. When a company spends a huge amount of money to promote its brand, the promotion is expected to be effective. Identifying the target population and figuring out the best method to reach them is important for a promotion to turn effective. The selection of suitable media such as TV, radio, or Internet helps improve the image of the product or service.
- The promotion should empower something. If one promotes a person in front of others, that person is empowered with a better image. In the case of a product promotion, it should be empowered with better ratings and reviews. In the context of a book release, the book needs other to say good things about it to be empowered.
- The promotion should raise awareness about something steadily. Progression means a gradual and steady promotion over a period of time. For example, a TV advertisement clip was repeatedly shown over several months should progressively increase sales.

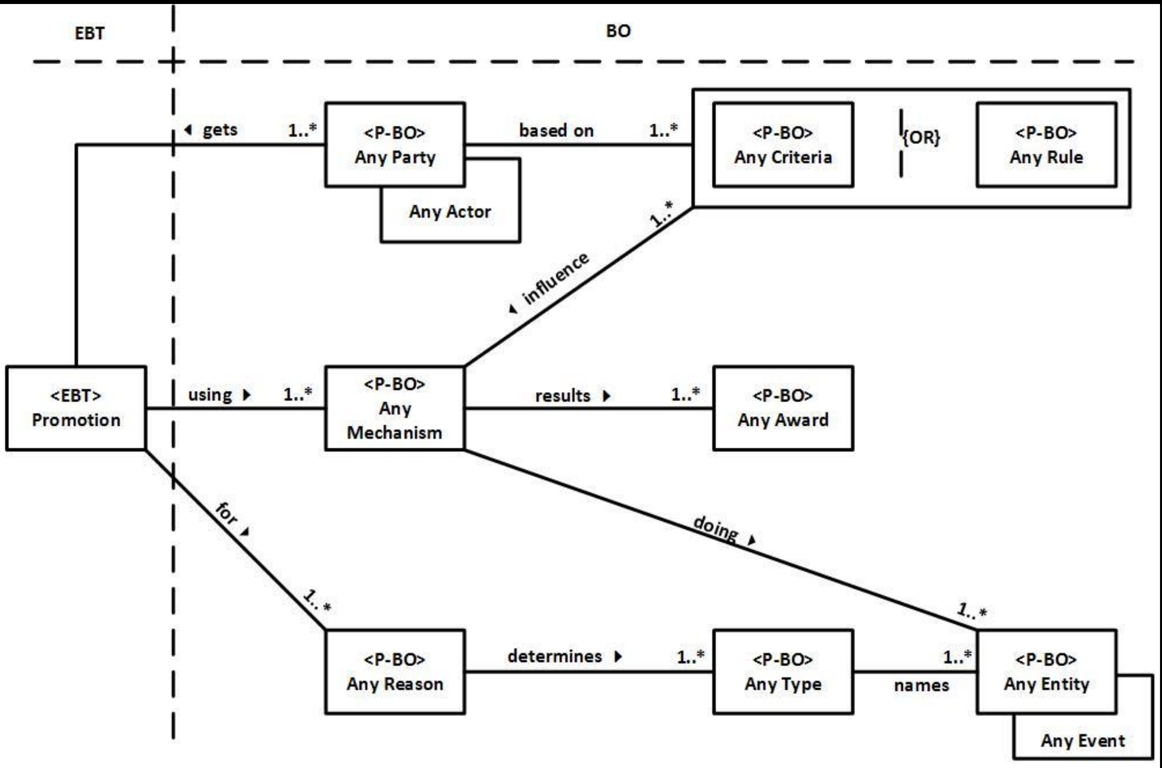


Figure 51. The class diagram of the promotion SAP.

Class diagram description:

- One or more AnyParty/actor gets promotion based on one or more any criteria/rule.
- Promotion happens for one or more AnyReason using one or more AnyMechanism.
- AnyMechanism results in one or more any award.
- AnyReason determines one or more AnyType.
- AnyType names one or more AnyEntity/event.

Summary. As a generic term, promotion is active word that has a wide usage in our lives. In the context of creating a pattern template, the most important requirement is to identify the most appropriate enduring theme. It is possible to extract it only through sophisticated evaluation of different terms for promotion. This chapter has succeeded in identifying the main theme for developing this template. The mid-size template provided in this paper exhibits how core knowledge of AnyPromotion can be used for creating numerous applications. It also interprets an SDP for AnyPromotion in an effective manner. This model can also be extended to identical contexts while reusability of the pattern could be a revealing fact.

Selling SAP: Pattern Documentation

Pattern name: Selling SAP. Selling is an EBT. Selling means to exchange something for money. Selling can be an activity in exchange for cash or goods and services. Other names for selling are trade, auctioning and market ("Selling," 2016). The reason for choosing the term selling is because it is a more generic term than others for the proposed pattern. This lead to an SAP.

Context. Selling has applications in various domains, such as real estate, services and products, gas and oil, and health service. Types of selling are team selling, personal selling, and relationship selling. The following are some of the contexts where we can apply selling as SAP.

- Social Selling: Social selling (Selling) is a type of sale, where products are sold on social network platforms like Facebook (AnyParty). Consider a situation, where a used car dealer (AnyParty) posts pictures and other details about a car

(AnyEntity) for sale on a buy and sell Facebook page. Other members of the page, who are interested in buying a used car, be able to see the information about the car and contact the dealer for more details. The dealer uses social networking (AnyMechanism) as a technique for selling the car because more customers can be targeted (AnyReason) through Facebook. Social selling is also more economical than other means of sale. This type of online marketing (AnyStrategy) helps the dealer increase sales. New offers may be introduced such as new tires (AnyDeal), if purchase is made before a specific date. Facebook belongs to social connections (AnyType).

- Cross-selling: Cross-selling (Selling) is a type of selling, where an extra product or service is sold to an existing customer (AnyParty). This type of selling mostly happens in banks and other financial (AnyType) institutions. Consider a scenario where a laptop seller (AnyParty) offers his customers to buy a mouse after they purchase a laptop (AnyEntity). The seller could introduce a discounted price (AnyDeal) for buying both the laptop and the mouse together and for buying them separately. Such an offer could entice the customer to take the deal, which in turn increases the profits for the seller. The suggesting (AnyStrategy) of a related product to a customer is the main idea behind cross-selling. The approach used in cross-selling is that the seller would recommend (AnyMechanism) an additional product to the customer. Cross-selling helps in expanding business relationships (AnyReason).

Functional requirements.

- Selling represents the exchange of a product or service for money. It has attributes such as itemOfSale, value, and availability. It has operations such as exchange(), transfer(), offer(), and deal().
- AnyParty class represents a country, political party, government, and persons belonging to an organization. AnyParty generally has a name, location, and phoneNumber. It have operations such as promote(), buy(), sell(), and trade().
- AnyActor represents someone or something that is involved in a selling activity. It has attributes such as name, id, and type. It has operations such as collectMoney(), interact(), suggest(), and recommend().
- AnyMechanism represents the method used for selling. It has attributes such as name, context, description, application, and status. It has operations such as execute(), attach(), detach(), and activate().
- Every sale has AnyReason and this class represents the reason for selling. It has attributes such as description, proof, and justification. It has operations such as conclude(), examine(), and resolve().
- AnyType determines the type which AnyReason class determines. It has attributes such as property, id, interfaceList, and name. It has operations such as change(), operateOn()classify(), resume(), and label().
- AnyDeal represents the AnyDeal which AnyParty or AnyActor wants during selling and can be achieved through AnyMechanism. It has attributes such as

agreement, amount, and degree. It has operations such as negotiate(), bargain(), and doBusiness().

- AnyStrategy represents is supported by AnyMechanism of selling. It has attributes such as name, technique, applicability, and context. It has operations such as combine(), design(), and prepare().
- AnyEntity represents the entity that is involved in selling and is related to the want of AnyParty or AnyActor. The attributes can be id, name, and status. Its operations can be to update(), relationship(), and type().
- AnyEvent also represents the event that is involved in selling and is related to the want of the party or actor. An event can be a service, which is sold to customers for a price. It has attributes such as name, occasion, type, and outcome. It has operations such as appear(), happen(), and arrange().

Non-functional requirements.

- Selling should be practical and doable. Selling something should be feasible, which means one should be able to sell that product or service. One cannot sell liquor to minors at a high school. It is not doable. The government prohibits the sale of drugs online, so an e-commerce website like Amazon, cannot sell cocaine on its website. The sale of cocaine on Amazon is not doable. Similarly, every sale of a product or service should be doable.
- : A product or service needs to be available to be sold. When we want to sell a service such as a cab service in San José, we can sell that service only in San Jose. One cannot sell that service in New York because the service is not available.

Hence, a product or service should be available in order to sell it. Therefore, selling can be based on the availability.

- When we sell something for a purpose, the commodity or service sold shall serve the purpose or should be relevant to the purpose. Consider a situation, where a grocery store owner is selling the spices used in Indian food at his store, which is located in a town where Indians do not live. There are slim chances making any profit by selling those spices in such a location. Hence, the product or service being sold should be relevant.
- Selling should be effective in order to sell more. Ineffective selling techniques not only result in reduced sales, but it also spoils the chances of selling in the future. Effective sales strategies are the main goals of any business.
- Valuable selling means selling something that is valuable to customers. Valuable selling techniques are sophisticated, and they need advanced training to succeed. Selling should also result in higher returns on money and investments. Valued selling can result in more voluminous turnover than selling low-value items.
- When someone sells something, it should be measurable by numbers, value, and turnover. One should be able to measure a person’s selling ability regarding rate of return on investment and overall profit. The measure is an invaluable metric that can help with selling activity.

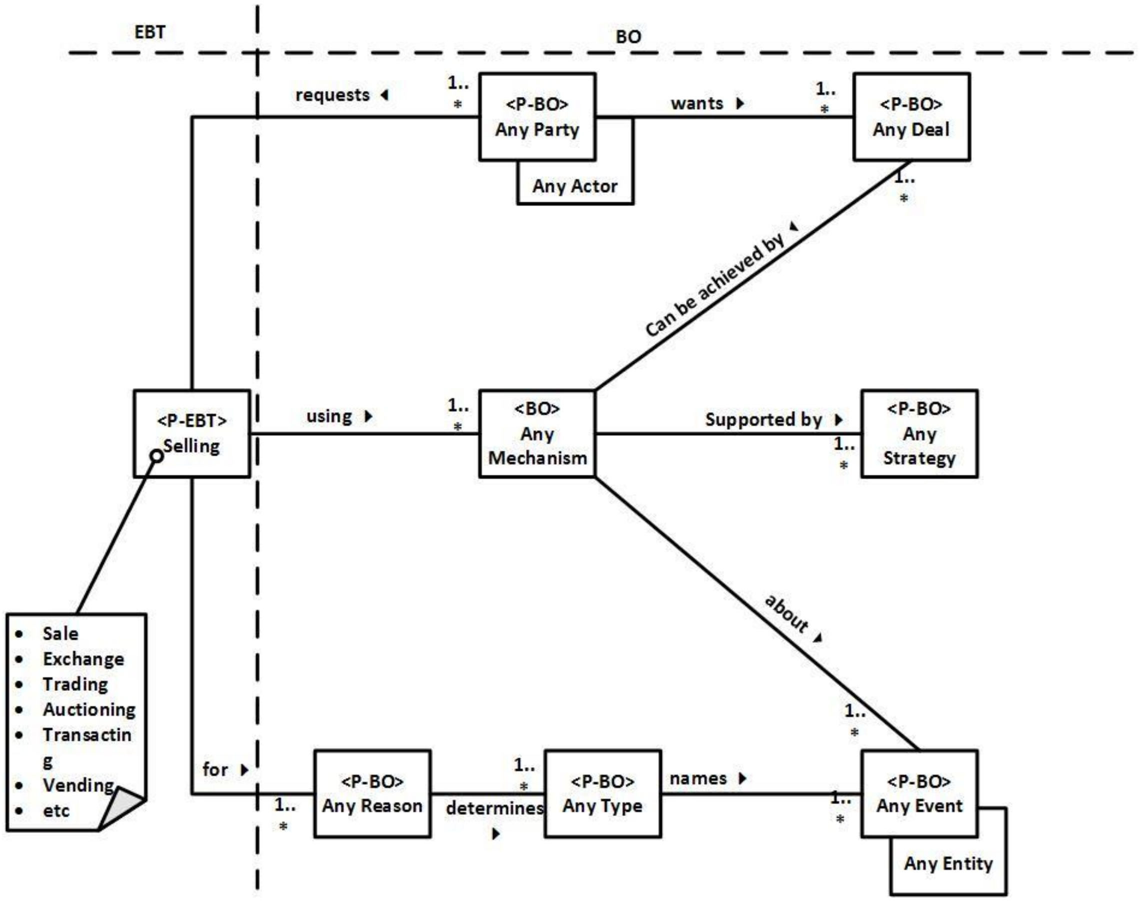


Figure 52. The class diagram of the selling SAP that includes EBTs and BOs.

Class diagram description.

- AnyActor or AnyParty requests a sale.
- AnyMechanism is used to sell for AnyReason.
- AnyReason determines AnyType.
- AnyType names AnyEntity or AnyEvent.
- AnyMechanism is about AnyEntity or AnyEvent.
- AnyMechanism can achieve AnyDeal.
- AnyParty or AnyActor wants AnyDeal.

Summary. The template provided in this chapter exhibits how the core knowledge of selling can be used for creating numerous applications. Being a general English term, selling can give rise to a pattern template that can help us identify the core theme for the word, and use it to create a solid and workable template. It also interprets SAP for selling in an effective manner. This model can also be extended to identical contexts, where reusability and robustness of the pattern itself are very big contributions.

Chapter 6: Future Work and Conclusion

Rapid globalization and competitive market scenarios have forced manufacturers and service providers to offer new and better products and services from time to time. However, in an attempt to make more profits, they often use unfair trade practices, which impacted consumer satisfaction. Customer satisfaction represents the degree to which a product or service has reached the expectations of a consumer. When a consumer experiences disappointment concerning a product or service, he or she usually files a complaint. A complaint is either formal or informal. It is regarded as a voice raised by a consumer against unfair treatment by the seller. According to the law, a consumer is also protected from being abused by businesses through consumer rights. Consumer rights are a way of protecting consumers and offering them appropriate action to take against sellers within the boundaries of the law. It is extremely necessary for a consumer to know his or her rights to make an effective complaint.

Introduction. Software patterns are widely used to play an important role in enhancing the standard of software systems. They are designed for reducing development time and cost efforts. However, using these patterns has been known to introduce issues that considerably cut back the soundness, robustness, and reusability of the software system. The SSM based design methodology is a new method for making software patterns and creating solid results in extremely stable, reusable, and cost-efficient software systems. The SSM uses standardized UML tools to specify, visualize, construct, and document components of a software pattern. It utilizes industry standard graphic notation to make visual models for any software package.

Based on the concepts of SSM, this thesis presented nineteen unique models for consumer complaints and protection in the form of stable analysis and design patterns. These patterns are designed in order to help developers build reliable software systems dealing with consumer related issues. All the patterns are generic in nature, which makes them applicable to all types of applications within the consumer complaints and protection concept. The reason for such extensive applicability of the patterns is that they focus only on the core knowledge during the inception phase of the solution. The stable patterns Thus, serve as a strong foundation for developing robust software applications. Furthermore, with these patterns, the existence and behavior of the problem statement can be understood explicitly (Goverdhana & Fayad, 2004, p. 4).

The SSM patterns can also be used to build applications that allow consumers to register complaints about any product or service. Such an application can be a simple way to reach out to complaint agencies for solutions, without much of an effort on the part of the consumer. Also, a combination of all of these patterns can be used to build an engine to deal with every aspect of consumer complaints and protection. The following methodology be utilized to put together the engine:

- Determine the goal and capabilities of the proposed engine.
- The problem area is divided into multiple patterns. Each pattern has components that form a knowledge map that is independent of any application logic.
- Then find the EBTs and BOs for those patterns. Different scenarios need to be studied for each pattern to find the stable core logic that can be expressed as the EBTs and BOs.

- Next, build a database for each system goal using constraint databases.
- The final task is to put all these patterns together to develop the engine.

Future Work. Due to the specific and rigid nature of traditional consumer complaints and protection software applications, they often fail to address user concerns adequately. Therefore, there is a need for a standard platform that can factor in any changes to the original architecture of the application while safeguarding user interest. Under such a scenario, the stable analysis and design patterns provided in this thesis would come into picture. The real-time amalgamation of these patterns would help build efficient software to deal with consumer rights and grievances. It can be used to not only educate consumers about their rights and responsibilities, but also to assist them in registering their complaints for any loss.

Social networking has changed the way consumers look at products and services. Consumers have now begun trusting peer recommendations more than the commercial advertisements. All types of businesses, small, medium or large, have started to use social media to reach out to their customers. In other words, with the rise of social media, the idea of community has changed radically. Social networks are not just about the people around us; they are about people who share interests. Therefore, we envision consumer complaint and reporting as a social network and content management system. The social network named ‘Aeeh.net’ help ease the process of grievance reporting and handling through a simple web-based user interface.

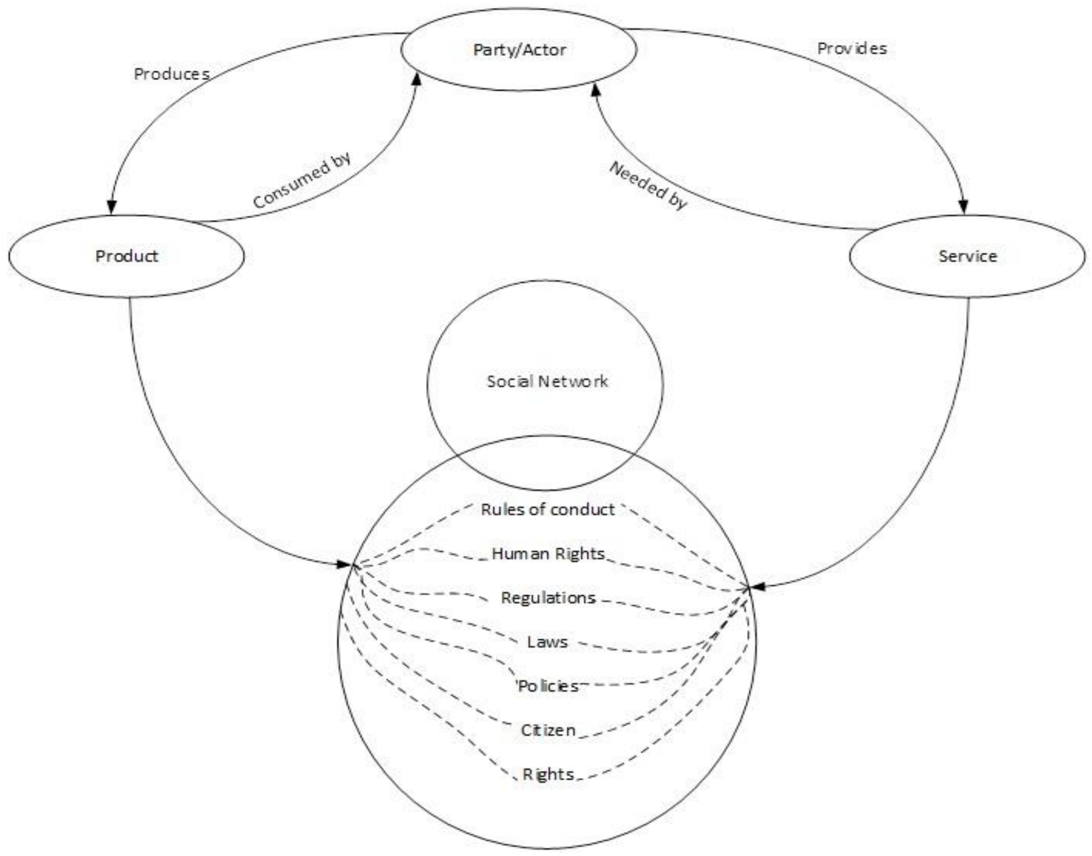


Figure 53. Aeeh.net – High-Level Overview

The underlying idea of creating Aeeh.net itself is a big contribution in the domain of consumer products and services reports. This network eventually protects consumers worldwide from dubious and unsubstantiated claims made by sellers. Figure 6.1 gives a high-level overview of the proposed social network. This social network is proposed to be built using the PHP scripting language, phpFox. This is a prominent framework and one amongst the several social networking packages that are commercially available on the market for those wanting to develop a social networking application. It provides software developers full command over different web development control options like website layout, making exclusive looks and feels, customizing options and ensuring high application performance.

The goal of Aeeh.net is to create a convenient platform for the consumers worldwide to express their feeling about the product or service. The members of this social network can complain here about the products or services with which they are dissatisfied. This would help countries around the world, especially where consumers are forced to use inappropriate and inferior products and services but cannot do anything about it. Customers can protect themselves by evaluating and providing feedback for products and services before planning to buy them. Based on such user feedback, Aeeh.net would furnish an in-depth evaluation report on the product or service chosen. This way, the network would expose any inappropriate or inferior products by identifying and tagging them. Furthermore, members would post their complaints in different formats such as digital proof like images, videos, voice recordings, and documents. By studying this, other could determine the reputation and authenticity of a product, seller, or service provider.

Conclusion. Consumer complaints and protection is a sensitive area that needs a immediate attention against increasingly unfair business practices. Although CRs make sure that the consumer’s voice is heard by providing a mix of advocacy and advice, their primary goal is to report the facts and experiences about what experts claim. Therefore, they lack user feedback-based reviews about products or services. Currently only some government agencies and third parties provide services to the customers for filing complaints about the seller. In many countries, government policies still do not allow shoppers to enjoy economic resources.

This thesis is a giant step toward addressing this issue by proposing an easy to use social networking based application. This application is intended to serve as an engine to deal with user concerns about a product or service and, at the same time, protect them against unfair practices. Such an engine can be built upon the foundations of the SSM concepts. Technically, the stable analysis and design patterns are very important aspects to strengthen the quality of any software product. The patterns designed by using the SSM have turned out to be reusable, stable, repeatable, robust, traceable, and cost effective. The traditional models provide a solution with reference to a single context, while the SSM patterns are easy to apply in various other scenarios in related context. The SSM based applications can be modeled for unlimited solutions just by hooking them up with the required concrete objects. This thesis illustrates how the core knowledge of the consumer rights protection topic can be used for building numerous but robust applications that promote scalability, extensibility, interoperability and flexibility. Although creating high-quality software applications with exhaustive capabilities and easy-to-do upgrades has been a difficult task to perform, this is where the stable analysis and design patterns would help developers to develop a number of stable applications dealing with consumer rights.

References

Advice. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Appraisal. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Beekman, V. (2008). Consumer rights to informed choice on the food market. *Ethical Theory and Moral Practice*, *11*(1), 61-72.

Cerri, S. (2000, August). *Effective communication skills for engineers.* Paper presented at the meeting of the Engineering Management Society, Albuquerque, NM. doi:10.1109/EMS.2000.872578

Commitment, (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Complaint. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Compliance. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Dakof, G. A., & Taylor, S. E. (1990). Victims' perceptions of social support: What is helpful from whom? *Journal of Personality and Social Psychology*, *58*(1), 80.

Daniel, C. N., & Berinyuy, L. P. (2010). *Using the SERVQUAL model to assess service quality and customer satisfaction* (Master's thesis). Retrieved from DiVA. (35008)

Davidow, M. (2003). Organizational responses to customer complaints: What works and what doesn’t. *Journal of service research, 5*(3), 225-250.

Deed. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Egan, K. A., & Arnold, R. L. (2003). Grief and bereavement care: With sufficient support, grief and bereavement can be transformative. *The American Journal of Nursing, 103*(9), 42-52.

Fayad, M. E. (2002). Accomplishing software stability. *Communications of the Association of Computing Machinery, 45*(4), 111-115.

Fayad, M. E. (2002a). How to deal with software stability. *Communications of the Association of Computing Machinery, 45*(4), 109-112.

Fayad, M. E. (2015). *SDPs for software and systems.* Boca Raton, FL: Auerbach Publications.

Fayad, M. E., & Altman, A. (2001). An introduction to software stability. *Communications of the Association of Computing Machinery, 44*(9), 95-98.

Fayad, M. E., & Hamza, H. (2003, September). The AnyAccount pattern. In *Proceedings of Pattern Language of Programs.* Conducted at the meeting of PLoP , Monticello, IL.

Fayad, M. E., & Hamza, H. (2004, August). The trust analysis pattern. In the *Proceedings from The Fourth Latin American Conference on Pattern Language of Programs*, Porto das Dunas, Ceara, Brazil.

Fayad, M. E., Hamza, H., & Stanton, V. (2004, August). The trust analysis pattern. In the *Proceedings from The Fourth Latin American Conference on Pattern Language of Programs,* Porto das Dunas, Ceara, Brazil.

Fayad, M. E., & Singh, S. K. (2011). Call for papers: Pattern Languages: Addressing the challenges (PLAC). *Wiley Journal on Software: Practice and Experience. 41*(1), 129-130. doi 10.1002/spe.1041

Fayad, M. E., Sanchez, H. A., Hegde, S. G., Basia, A., & Vakil, A. (2014). *Software patterns, knowledge maps, and domain analysis*. Boca Raton, FL: Auerbach Publications.Fayad, M. E., & Wu, S. (2002). Merging multiple conventional models in one stable model. *Communications of the Association of Computing Machinery, 45*(9), 102-106.

Fayad, M. E., & Wu, S. (2002). Merging multiple conventional models in one stable model. *Communications of the Association of Computing Machinery, 45*(4), 5.

Fayad, M. E., Rajagopalan, J., & Ranganath, A. (2003, October). AnyLog stable design pattern. Paper presented at the *IEEE International Conference on Information Reuse and Integration,* Las Vegas, NV. doi: 10.1109/IRI.2003.125466

Fernandez, E. B. (2005, May). Security patterns and secure systems design using UML. In the *International Conference on Enterprise Information Systems*, Miami, FL, USE.

Fernandez, E. B., Yuan, X., & Brey, S. (2000). Analysis Patterns for the Order and Shipment of a Product. In thethe meeting of the PLoP , Monticello, IL.Fowler,

M. (1997). *Analysis patterns: Reusable object models*. Boston, MA: Addison-Wesley Professional.

Goverdhana, R., & Fayad, M. E. (2004, November). Any Transaction SDP. In *Information Reuse and Integration, 2004.* Proceedings from *the 2004 IEEE International Conference*, 54-59.

Guideline, (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Gratifiction. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Hamza, H. S. (2002). Toward stable software analysis patterns. e *Companion of the 17th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications,* 110-111.

Hamza, H. S., & Fayad, M. E. (June, 2002). Model-based software reuse using SAPs. *ECOOP.*

Hamza, H. S., & Fayad, M. E. (2003). The negotiation analysis pattern. Paper presented at the *Europlop.*

Hamza, H. S., Mahdy, H. A., Fayad, M. E., & Cline, M. (2003). Extracting domain-specific and domain-independent patterns. Paper presented at the *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications:ACM,* 310-311.Howells, G. G., & Weatherill, S. (1995). *Consumer protection law*, 96, 4. Dartmouth.

Judgment, (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Klein, D.B. (4 May, 2015). "Consumer Protection." *The Concise Encyclopedia of Economics.*2008. Library of Economics and Liberty. Retrieved from: http://www.econlib.org/library/Enc/ConsumerProtection.html

Lai, Y. K., Lai, Y. F., & Lin, J. W. (May, 2012). High-quality view synthesis algorithm and architecture for 2D to 3D conversion. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium*, 373-376.

Liccardi, I., Bulger, M., Abelson, H., Weitzner, D. J., & Mackay, W. (2014). Can apps play by the COPPA Rules?. In *Privacy, Security and Trust (PST), 2014 Twelfth Annual International Conference*, 1-9.

Liu, Q., & Karahanna, E. (2015). An agent-based modeling analysis of helpful vote on online product reviews. Paper presented at the *System Sciences (HICSS), 2015 48th Hawaii International Conference on,* 1585-1595. doi:10.1109/HICSS.2015.192

Mahdy, A., & Fayad, M. E. (2002). AN SSM Pattern. In *Proc. of the 9 th Conference on Pattern Language of Programs (PLoP02), Illinois, USA.*

Miller, C. J., Harvey, B. W., & Parry, D. L. (1998). *Consumer and trading law: text, cases, and materials*. Oxford University Press, USA.

Model. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Need. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Nottage, L. (2004). *Product safety and liability law in Japan: From Minamata to mad cows*. Boca Raton, Fl: Routledge.

Oestreicher, L. (2007). Cognitive, social, sociable or just socially acceptable robots? Paper presented at the *Robot and Human Interactive Communication, 2007. RO-MAN 2007: the 16th IEEE International Symposium on*, 558-563.

Olds, B. M. (2000). Acceptable use policies and electronic mail: What are the frontiers? Paper presented at the *Frontiers in Education Conference, 2000. FIE 2000. 30th Annual*, *2* S3B/1-S3B/2 vol. 2.

Ownership. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Promotion. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Rate. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Review. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Selling. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Summers, D. (2003). Longman dictionary of contemporary English: (the living dictionary). Essex, U.K: Longman.

Support. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Turetken, O., Elgammal, A., van den Heuvel, W. J., & Papazoglou, M. P. (2012). Capturing compliance requirements: A pattern-based approach. Software, IEEE, 29(3), 28-36.

View. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Violation. (2016). *Merriam-Websters Online Dictionary* (11 ed.). Retrieved from www.merriam-webster.com

Appendix A

Appendix A provides four knowledge Maps:

A.1	CR Knowledge Map
A.2	Products Knowledge Map
A.3	Services Knowledge Map
A.4	Complaints Knowledge Map

A.1 The CR Knowledge Map

KM Name: CR

KM Nickname: None

KM Domain/Subject/Topic Description: Generally, CRs are a set of reports about different products and services that help in analyzing the good, bad and ugly of a product or a service. CRs include research about a product or service to give its merits and demerits. It also includes advertisements about any product or service. Consumer protection and complaint are subsets of CRs. Human Rights, which include consumer rights, protect the consumer from abusive business practices. For example, there is no doubt that a consumer must be given safe food for consumption, which means that a consumer has unconditional right toward safe food products. A complaint comes into picture, when any of the human rights are violated, thereby CRs helps in deciding the good, bad and ugly side of a product or a service.

EBTs/Goals: The below Table A1.1 describes the EBTs of CRs.

Table A1.1
EBTs of CR

EBTs/Goals	Description
Protection	When someone or something is protected from an unfavorable situation. Protection can be given to consumers through a set of protection laws. Protection clauses also help consumers to file complaints about erring business firms and sellers.
Avoidance	The act of avoiding someone or something from an unsavory situation.
Analysis	A careful examination of something in order to understand it better.
Compliance	A situation when someone obeys a rule, agreement or

	demand.
Inspection	A careful examination of something to find out more about it or to check for anything wrong.
Promotion	An activity planned to sell help sell a product or service that is being advertised.
Understanding	Being kind about other people’s problems.
Assessment	A calculation about the cost or value of something.
Improvement	The act of improving something like after sales service.
Justice	The system by which someone or something is judged in the courts of law.
Declaration	An important official statement about a specific situation.
Support	Approval, encouragement, and perhaps help for a person, plan or a strategy.
Recognition	The act of realizing and accepting that something is true and important.
Evaluation	A judgment about how good, useful or successful something is.
Judgment	An opinion that one forms especially after thinking carefully about something.
Advising	To tell someone what you think they should do especially when one knows more than they do about something.
Quality	How good or bad something is.
Research	An activity of finding information about something that one is interested in or need to know about.
Right	A behavior that is morally and ethically good and correct
Advertisement	A picture, a set of words, or a short film that is planned to persuade people to buy something.

Quality Factors: The quality factors are described in the Table A1.2:

Table A1.2
Quality Factors of CR

EBTs/Goals	Description
Preventable	To stop something from happening or stop someone from doing something.

Consistency	The quality of always being the same or having the same standards
Timely	Done of happening at exactly the right time.
Relevance	Directly relating to the subject or problem being discussed or debated.
Informative	Providing many useful facts or ideas
Understandable	Behavior or reactions that seem normal and reasonable because of the situation someone is in.
Convincing	Making someone believe that something is true or right.

BOs/Properties: The BOs of CR are described in the below table A1.3.

Table A1.3
BOs of CR

BOs/Capabilities	Description
AnyParty/Actor	AnyParty is the legal user of the system. AnyParty is classified into four types: buyer, seller, consumer forum, and organization.
AnyRule	A Rule is a lengthy statement defined according to the existing laws and regulations of a jurisdiction authority that highlights the governing procedure or controlling procedure within a particular system.
AnyType	A type is a category of things or issues distinguished by some common characteristic, attribute or quality.
AnyEntity/Event	An item that an organization must have in order to be able to conduct business. An entity can be a tangible or a non-tangible thing like specifications, service, plan, price etc. AnyEvent is a happening that has some or the other impact on the business. An event can be external as well as internal to the business.
AnyLog	A log is a file or a medium to record the events and outcomes that occur as a result of the business rule generation and standardization process.
AnyMedia	The means of communication, as radio, television, newspapers, and magazines, with wide reach and influence. A means for storing or communicating information.

AnyAdvise	An advice is a recommendation given to tell someone, the best thing to do in a certain scenario. In other words, Advise is a term used to signify suggestions, opinions and recommendations about a certain situation in different contexts.
AnyPolicy	A policy is a protocol or method of action to guide and determine decisions and achieve rational outcomes embracing the general goals and procedures in light of given conditions
AnySystem	A system is an organized entity that is created after including different working parameters that work together in unison for a specific purpose.
AnyGuideline	A guideline is an instruction or direction, which explains how a thing needs to be carried out.
AnyProtection	When someone or something is protected from danger or unforeseen circumstances. In the sales domain, it means protecting someone from a bad purchase or inferior purchasing decision.
AnyViolation	A violation is an action of doing something that is not allowed or illegal. Violation also means a deliberate infringement of a law or a set norm.
AnyData	AnyData refers to information regarding complaints made by a consumer to the seller.
AnyComplaint	Complaint is a written or spoken statement, expression or claim conveying a sense of displeasure toward a product, service and process
AnyProduct	AnyProduct is a something that is sold by a seller to a consumer. It could be a product or a service.
AnyService	AnyService is a specific type of help or work that is provided by a seller to consumer but not the one that involves producing products.
AnyCollection	Bringing things together. It is a set of identical things that are kept or brought together because they are attractive.
AnyCompensation	Money paid to someone because they have purchased something whose quality is bad or something they purchased which was different than the one that was advertised by a seller.

AnyConsequence	Anything that happens because of a result of particular action or a set of conditions.
AnyOutcome	It is a situation, when no one knows what it be until it actually happens. It is also the final result of a meeting or negotiation.
AnyImpact	The effect or influence that an event or situation has on someone or something.
AnyWarranty	A written agreement in which a seller selling something promise to repair or replace if a product breaks or fails to perform within a certain period.
AnyAgreement	A legal arrangement to do something that is made by two or more parties, sellers or businesses.
AnyAd	A set of words, images or a visual media that is created to persuade future buyers to buy a product or a service.

Knowledge Map (Core Knowledge): Each EBT is mapped to its corresponding BO in the below table A1.4:

Table A1.4
Knowledge Map of CRs

EBTs	BOs
Compliance	AnyParty, AnyActor, AnyOutcome, AnyAgreement, AnyMedia, AnyLog, AnyEntity, AnyEvent, AnyRule
Judgment	AnyParty, AnyOutcome, AnyType, AnyEntity, AnyEvent
Promotion	AnyParty, AnyActor, AnyEntity, AnyEvent, AnyRule

A.2 Product Knowledge Map

Km name: products

Km nickname: none

Km domain/subject/topic description: An article or substance that is manufactured or refined for sale.

EBTs/Goals: The below table A2.1 describes the EBTs of products.

Table A2.1
EBTs of Products

EBTs/Goals	Description
Ownership	The state or fact of being an owner
Applicability	That can be applied; relevant or appropriate
Essentiality	The quality of being essential; necessary

Quality Factors: The quality factors are described in the table A2.2:

Table A2.2
Quality Factors of Products

EBTs/Goals	Description
Usability	Fit for use; convenient to use
Innovation	Something new or different introduced
Trustworthiness	Reliable

BOs/Properties: The BOs of CR are described in the below table A2.3:

Table A2.3
BOs of Products

BOs/Capabilities	Description
AnyAuction	A public sale in which items of merchandise are sold to the highest bidder.

AnyFactory	A manufacturing center for producing goods and services. It is a source for manufacturing error free and quality products.
AnyClaim	It is a demand request made by a customer (AnyParty) to either replace or refund money owed to a defective or unsatisfactory product.
AnyAccount	The act of considering or including particular facts or details when making a decision or judgment about something.
AnyPresentation	The way in which something is said, offered, shown or explained to others.
AnyProject	A carefully planned piece of work to get information from something or to improve something.
AnyResource	To provide material or money for something.
AnyDistribution	The act of sharing things among people in a planned manner, or, supplying products and services to shops for selling.
AnyAccomplishment	Something successful or impressive that is achieved after a lot of effort and hard work.
AnyParty	It represents the party which provides or seeks support. It models all of the parties that are comprised in the support process. A party could be a person, organization, country, or political party.
AnyType	It represents the different types of support that can be provided or needed by an actor or a party.
AnyMechanism	It represents the mechanism that is used to perform the support operation.
AnyRule	An official instruction that mandates how things should be carried out or what should be allowed.
AnyDomain	An area of activity or knowledge especially one that a particular person or business deals with.
AnyPolicy	A policy is a protocol or method of action to lead and determine decisions and achieve positive outcomes that encompass general goals and procedures in the context of given conditions.

AnyMedia	The mode of communication, like radio, television, newspapers, and magazines, with wider geographical reach. It is also a mode for storing or communicating information.
AnyConstraint	It represents the details of the limitations or constraints of support process such as any external factors that can hinder the ongoing support operations.

Knowledge Map (Core Knowledge): Each EBT is mapped to its corresponding BO in the below table A2.4:

Table A2.4
Knowledge Map of Products

EBTs	BOs
Ownership	AnyClaim, AnyAccount , AnyParty, AnyProject, AnyAccomplishment
Applicability	AnyPresentation, AnyType , AnyMechanism, AnyRule , AnyPolicy, AnyResource
Essentiality	AnyProject, AnyParty, AnyRule, AnyPolicy, AnyResource

A.3 Services Knowledge Map

KM Name: Services

KM Nickname: None

KM Domain/Subject/Topic Description: Service is the action of helping or doing work for someone. The quality of service and customer satisfaction are very important concepts in any business. Quality of service is crucial because it it give way to a better customer satisfaction. Various applications involving services can be modeled using this knowledge map.

EBTs/Goals: The below table A3.1 describes the EBTs of CRs.

Table A3.1
EBTs of Services

EBTs/Goals	Description
Need	To have something or someone because one cannot do something without them.
Progression	Gradual process or change of development.
Utilization	To use some thing for a specific purpose.
Support	To help someone with something.

Quality Factors: The quality factors are described in the table A3.2:

Table A3.2
Quality Factors of Services

EBTs/Goals	Description
Usability	Something that is usable can be used.
Affordability	To have something or own something to buy or pay.
Evolving	To develop and change gradually over time that is usually longer.
Achievable	The capacity to achieve something that is usually successful
Sufficient	As much as is needed for a specific purpose
Intangibility	The state of feeling that is difficult to describe precisely
Perishability	The character of something that is likely decay or loose quality very quickly.
Variability	The character of something that may be different in different

	situations, so that someone is not too sure what may happen.

BOs/Properties: The BOs of CR are described in the below table A3.3:

Table A3.3
BOs of Services

BOs/Capabilities	Description
AnySkill	Proficiency with something
AnyProject	A proposal regarding a particular purpose
AnyPresentation	Exhibiting something to others
AnyParty	A person, organization, political party or country
AnyResource	Something that can be given when needed
AnyTask	Piece of work
AnyMilestone	An important point during any work
AnySchedule	A plan for doing something
AnyRule	Standard
AnyMechanism	A technique for doing something
AnyDomain	A field of knowledge
AnyCriteria	A standard on which decisions can be made
AnyType	Categorizing things
AnyRepresentation	Something that stands for another thing
AnyBrainstorming	Shared problem being solved by a group
AnyCommitment	Trusting someone or something
AnyLiaison	One that provides information to others

Knowledge Map (Core Knowledge): Each EBT is mapped to its corresponding BO in the below table A3.4:

Table A3.4

Knowledge Map of Services

EBTs	BOs
Need	Party, Resource, Project, Task, Milestone, Type
Obligation	Mechanism, Party, Commitment, Rule
Applicability	Representation, Criteria
Knowledge	Brainstorming, Skill, Domain,
Progression	Party, Milestone, Project, Schedule
Utilization	Party, Task, Resource, Skill,
Support	Party, Type, Mechanism, Liaison

A.4 Complaint Knowledge Map

KM Name: Complaint

KM Nickname: None

KM Domain/Subject/Topic Description: Complaint is an expression that a situation is disappointing.

EBTs/Goals: The below table A4.1 describes the EBTs of CRreports:

Table A4.1
EBTs of Complaint

EBTs/Goals	Description
Displeasure	It is a feeling of being annoyed or not satisfied with someone or something.
Dissatisfaction	Feeling of not being satisfied.
Disagreement	Approval situation in which people express differing opinions about something.
Violation	An action that breaks law or an agreement.
Grievance	A belief that one has been treated unfairly and unjustly.
Selling	The job and expertise of persuading people to buy something.
Guidance	Help and advice given to someone about some work or personal life.

Quality Factors: The quality factors are described in the table A4.2:

Table A4.2
Quality Factors of Complaint

EBTs/Goals	Description
Reasonable	Fair and sensible

Well-defined	Clear and easy to see or understand
Timely	A work or deed done and carried out the right time

BOs/Properties: The BOs of CR are described in the below table A4.3:

Table A4.3
BOs of Complaint

BOs/Capabilities	Description
AnyParty/Actor	A person or group involved in an enterprise; a participant.
AnyRule	Any standards rule for some procedure.
Any Complaint	Any expression for a situation that is disappointing.
AnyReason	The basis or motive for an action, decision, or conviction.
AnyType	A type is a category of things distinguished by some common characteristic or quality.
AnyEntity/Event	Something that is perceived, a noteworthy happening.
AnyLog	AnyLog that can be used to store files.
AnyMedia	The different ways of communication like radio, TV and newspapers.

Knowledge Map (Core Knowledge): Each EBT is mapped to its corresponding BO in the below table A4.4:

Table A4.4
Knowledge Map of Complaints

EBTs	BOs
Displeasure	AnyActor, AnyParty, AnyRule, Any Complaint, AnyReason, AnyType, AnyEntity, AnyEvent, AnyLog, AnyMedia
Selling	AnyActor, AnyParty, AnyReason, AnyType, AnyEntity, AnyEvent, AnyDeal, AnyStrategy,

	AnyMechanism.
Guidance	AnyActor, AnyParty, AnyRule, Any Guideline, Any Action, AnyReason, AnyEntity, AnyEvent

Appendix B

Appendix B has a list of:

B.1 SAPs (SAPs)

B.2 SDPs (SDPs)

B.1 SAPs (SAPs)

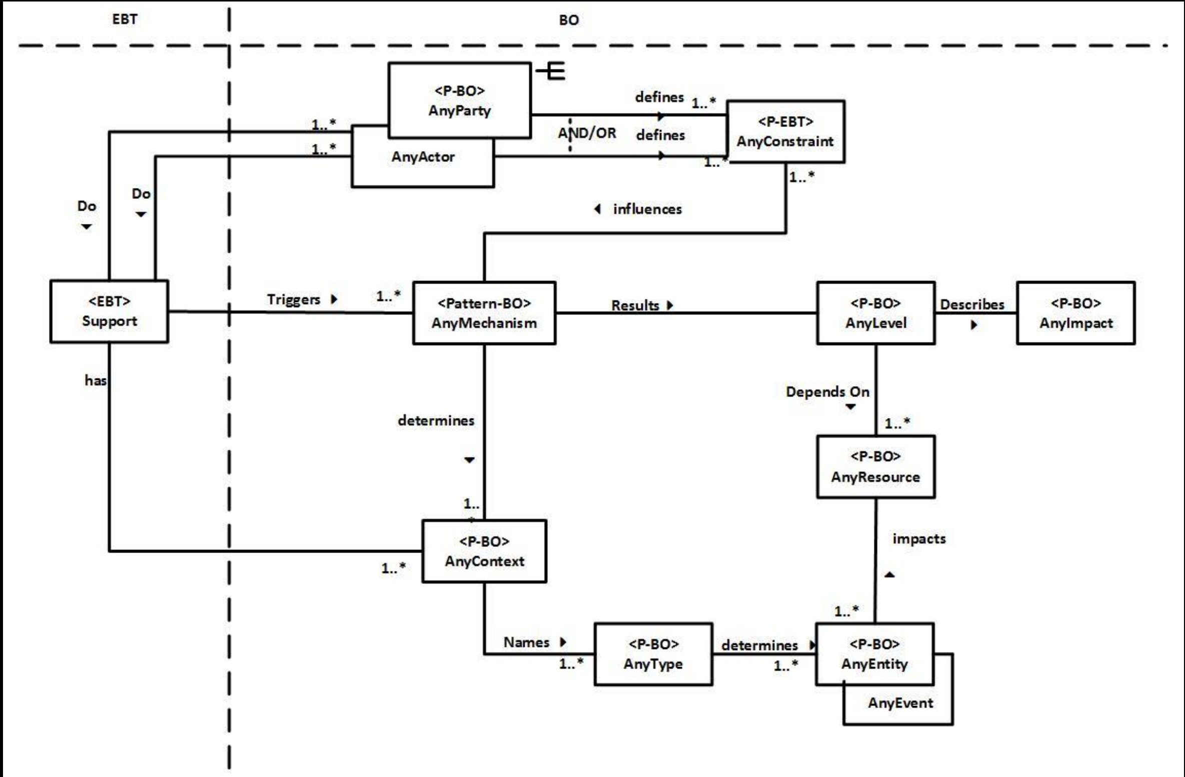


Figure B1.1: Support SAP

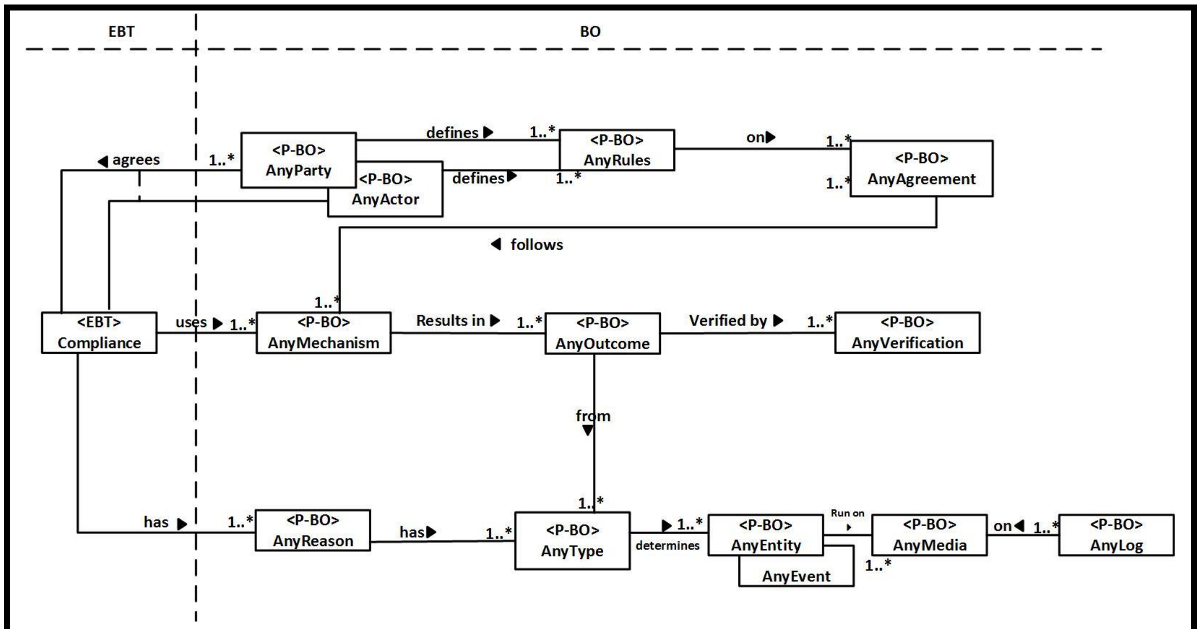


Figure B1.2: Compliance SAP

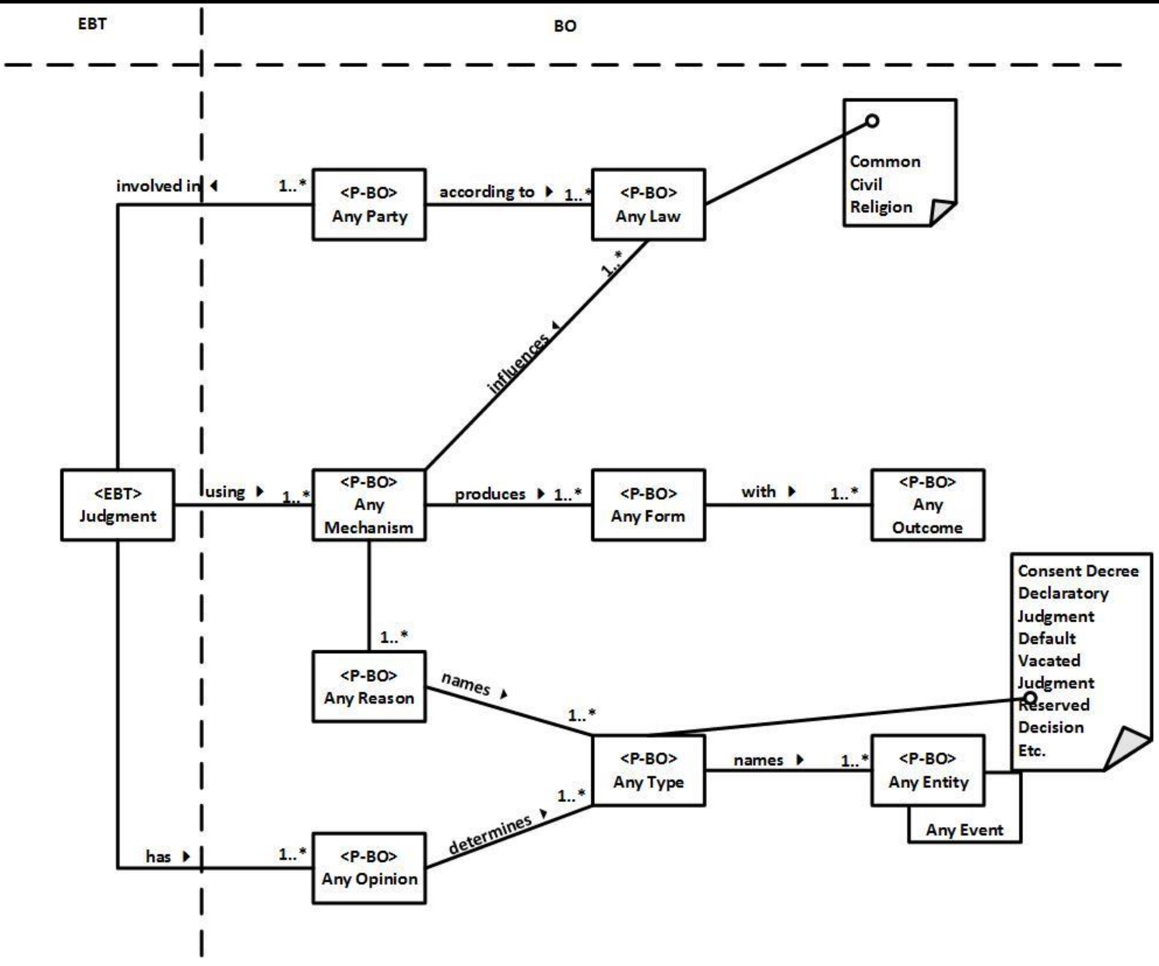


Figure B1.3: Judgment SAP

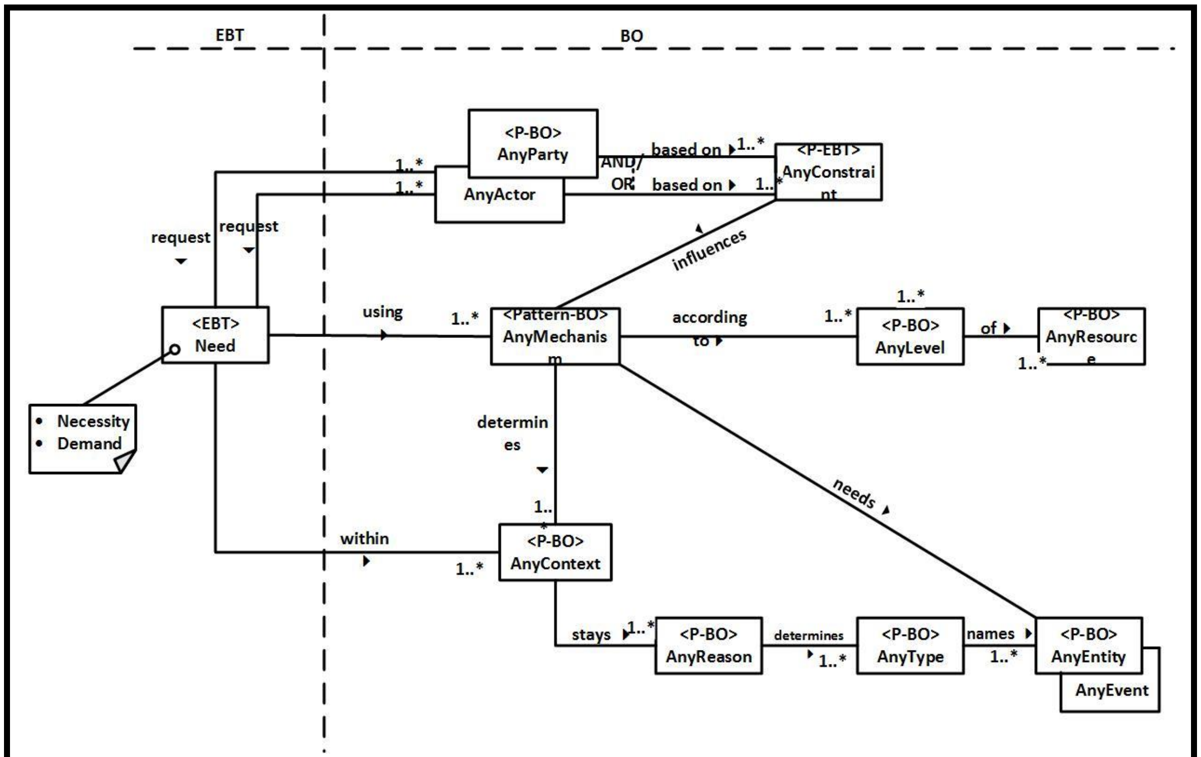


Figure B1.4: Need SAP

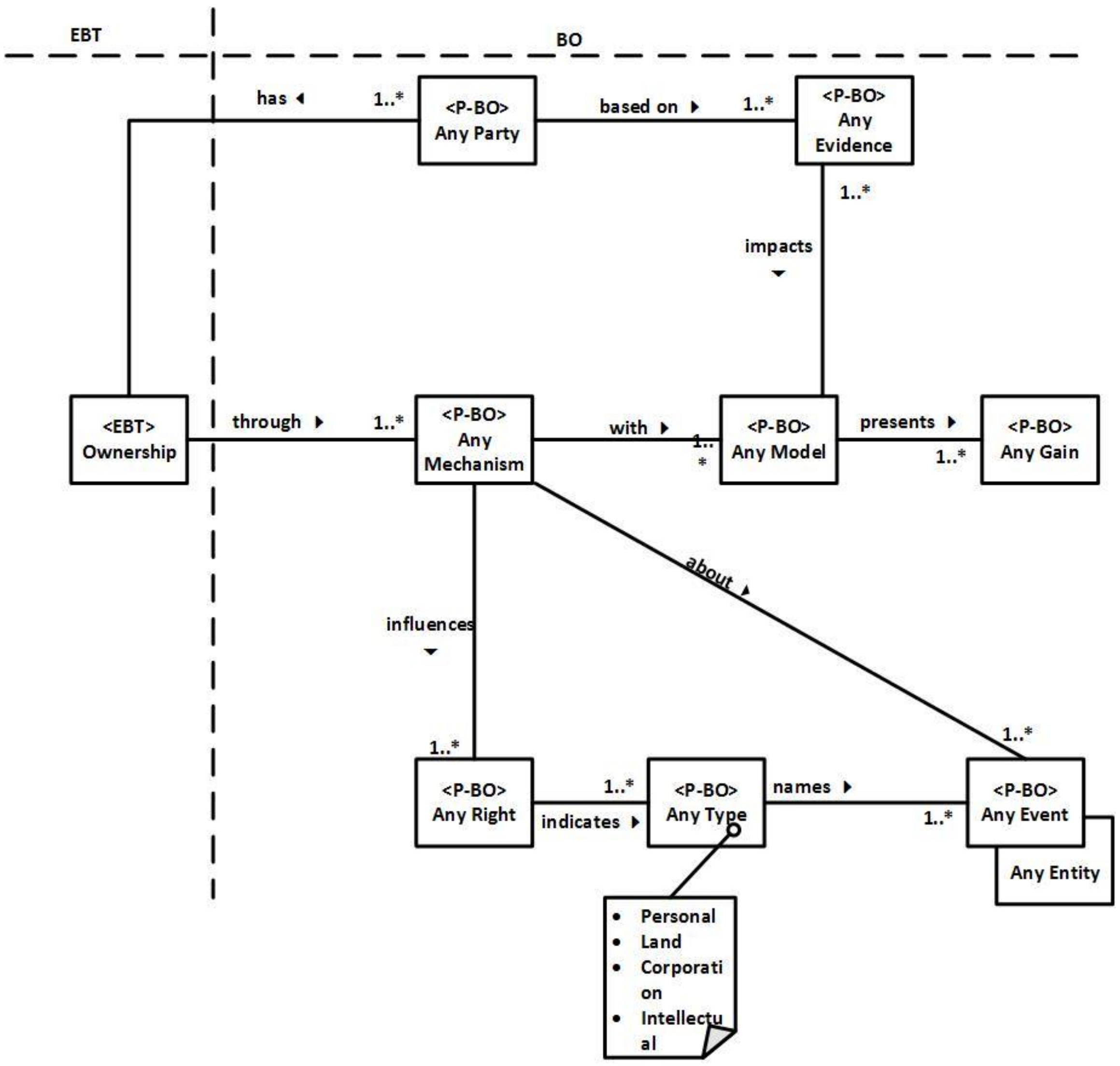


Figure B1.5: Ownership SAP

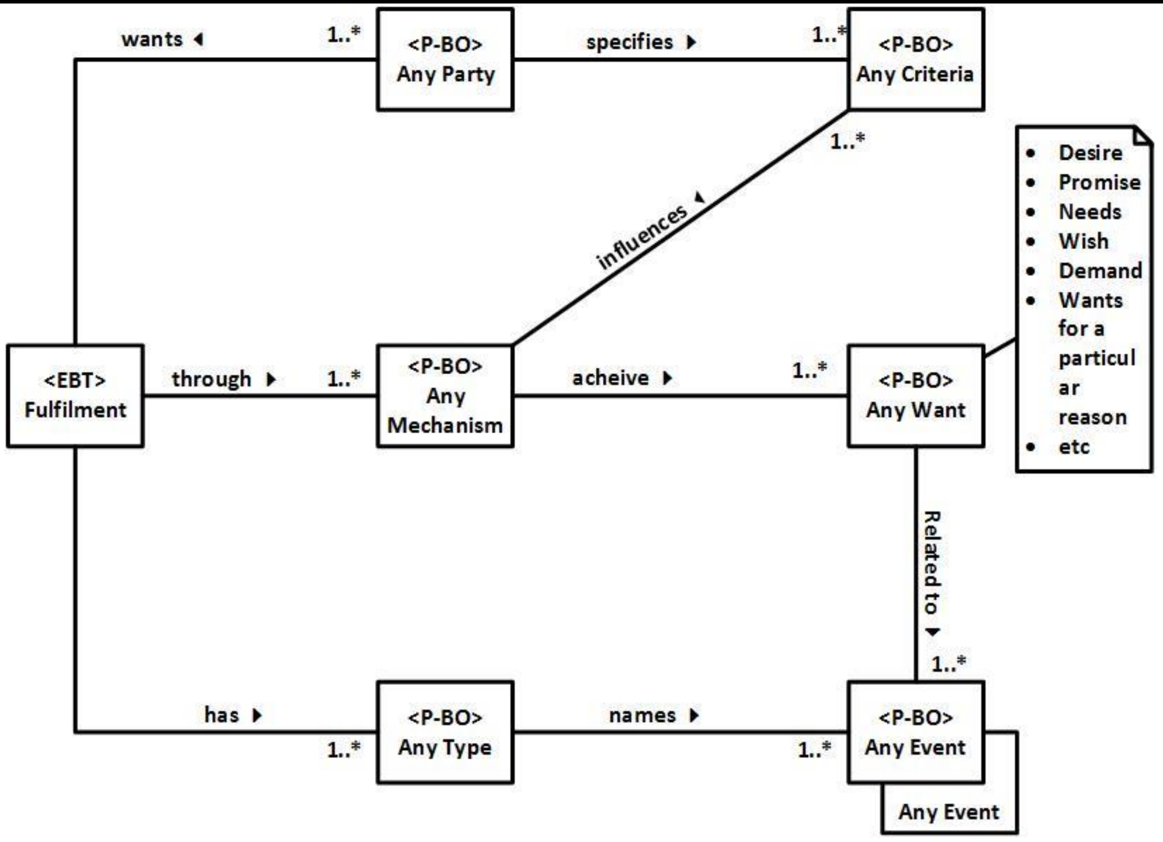


Figure B1.6: Fulfillment SAP

\

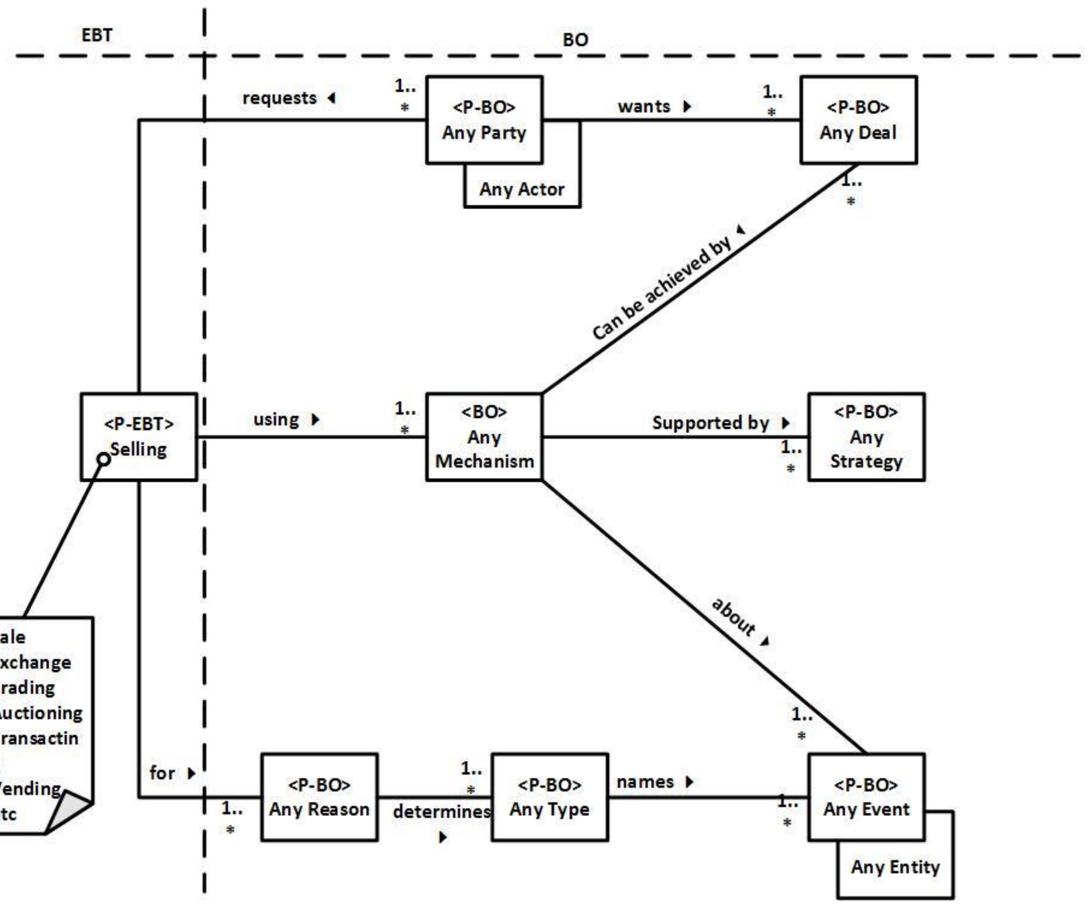


Figure B1.7: Selling SAP

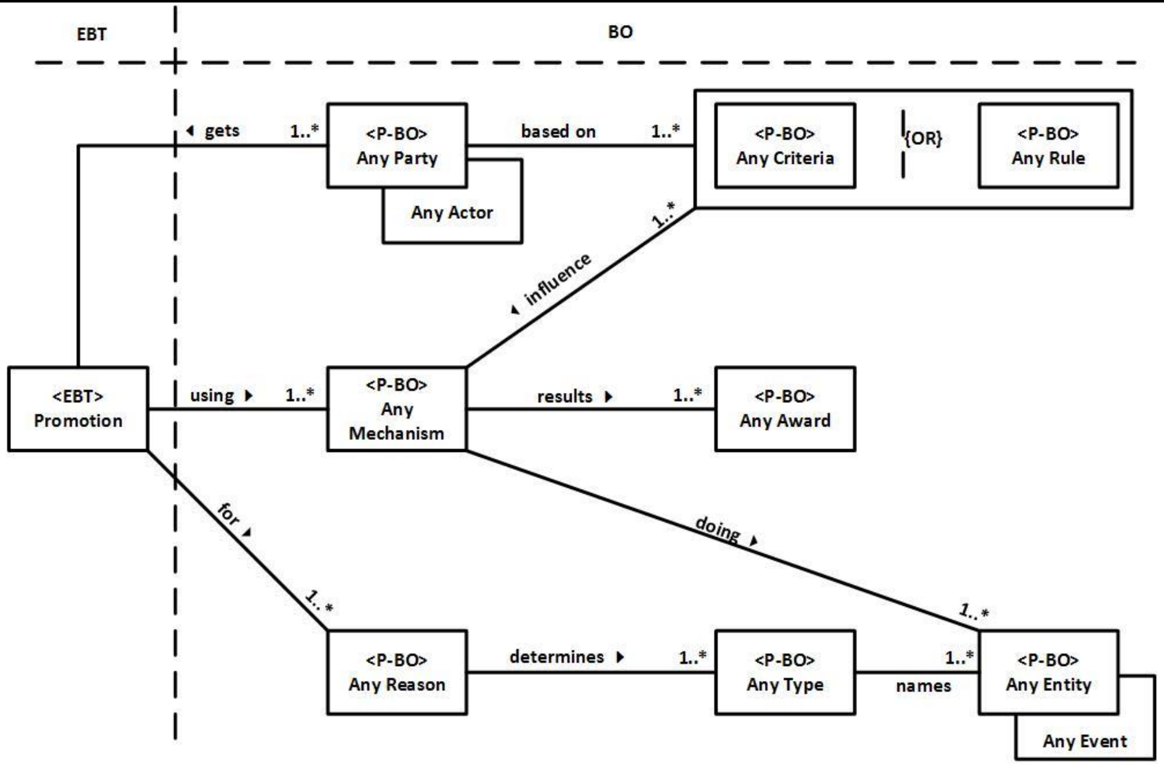


Figure B1.8: Promotion SAP

B.2 SDPs (SDPs):

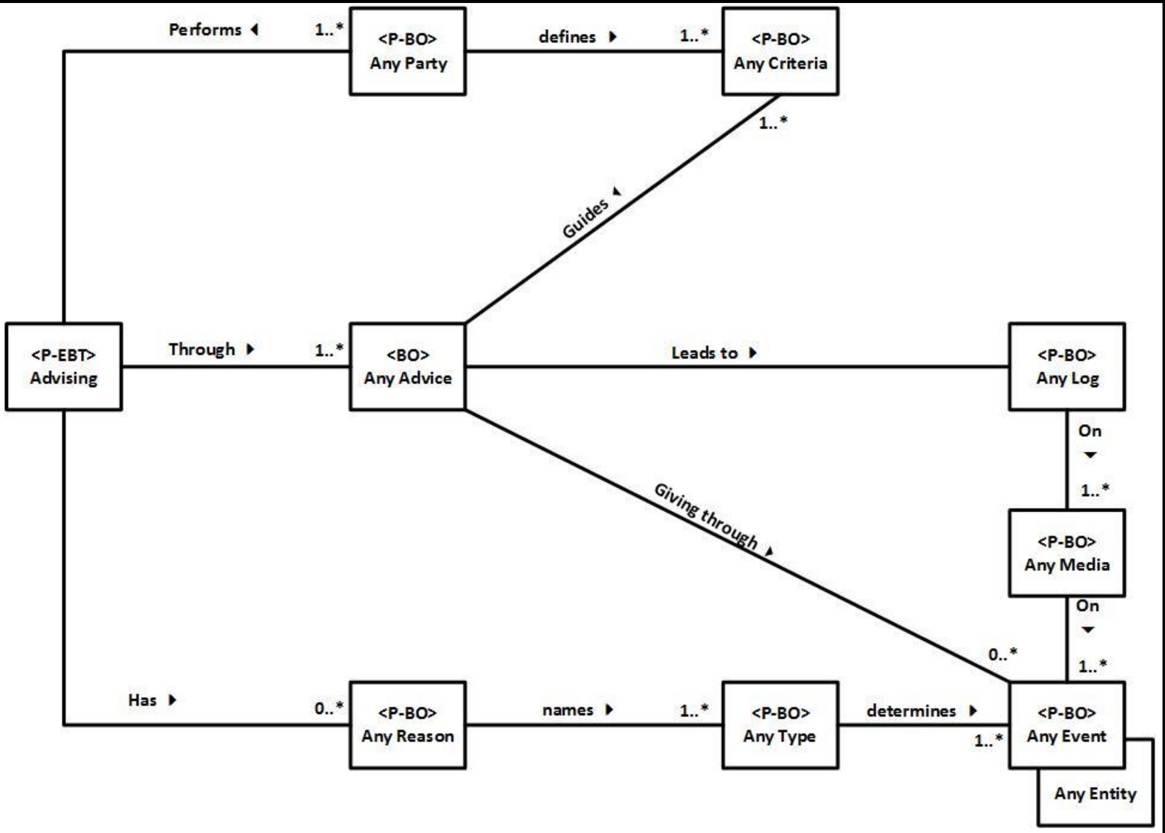


Figure B2.1: AnyAdvice SDP

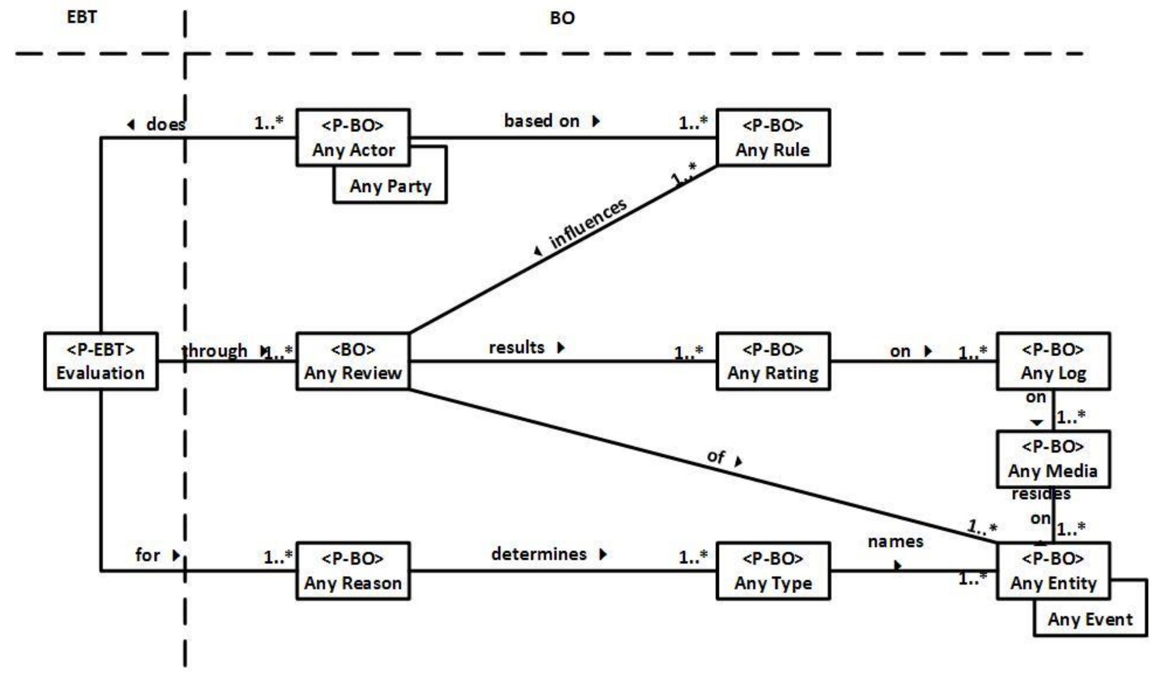


Figure B2.2: AnyReview SDP

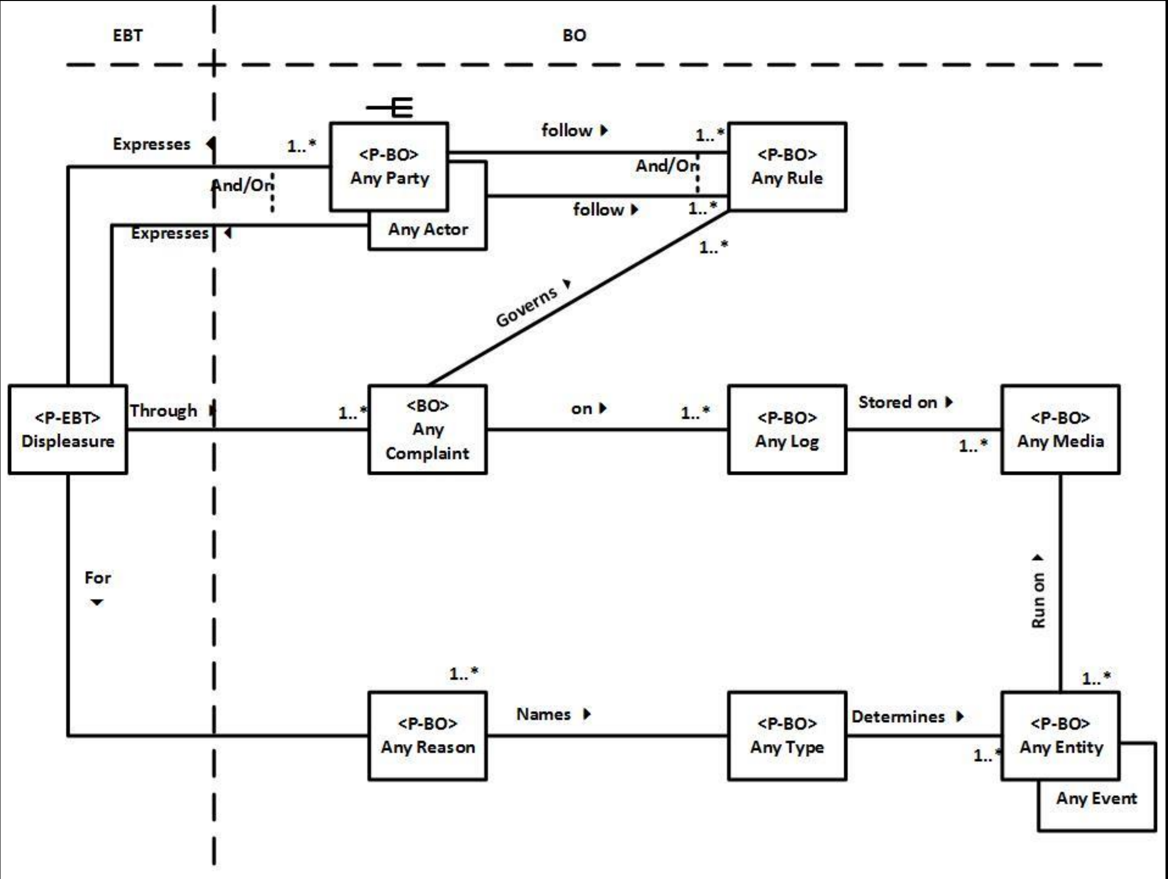


Figure B2.3: AnyComplaint SDP

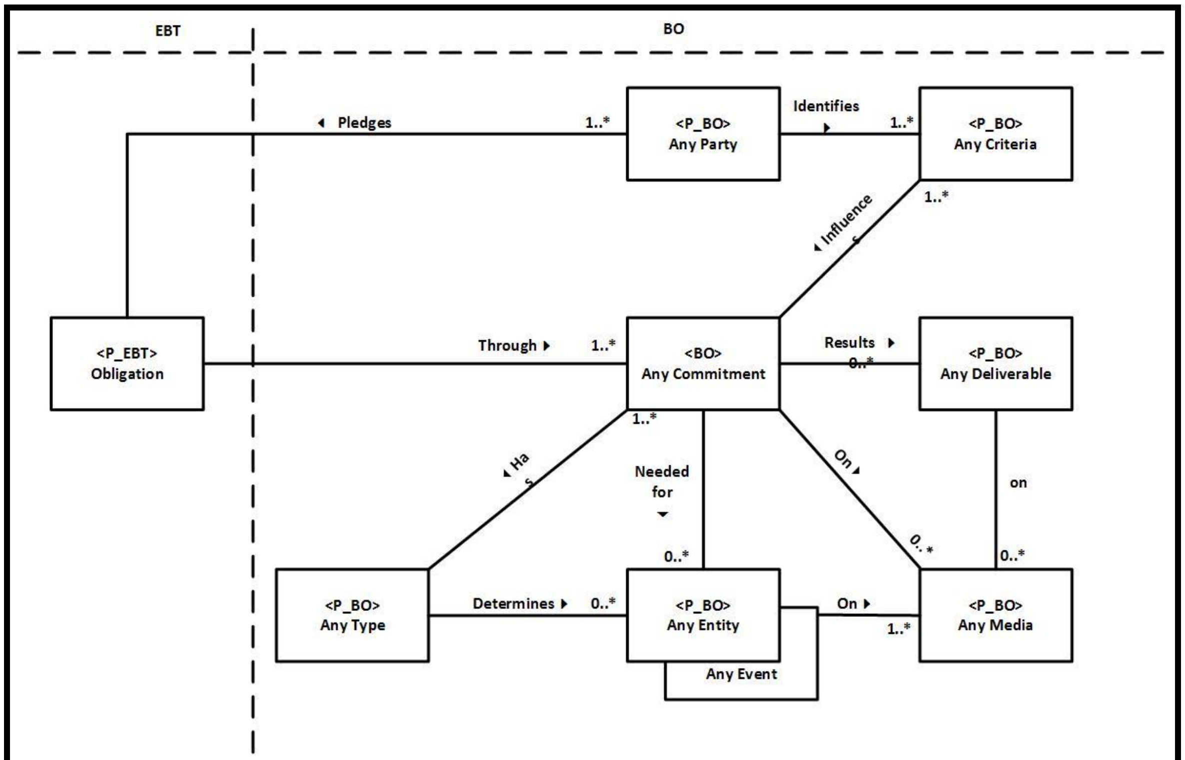


Figure B2.4: AnyCommitment SDP

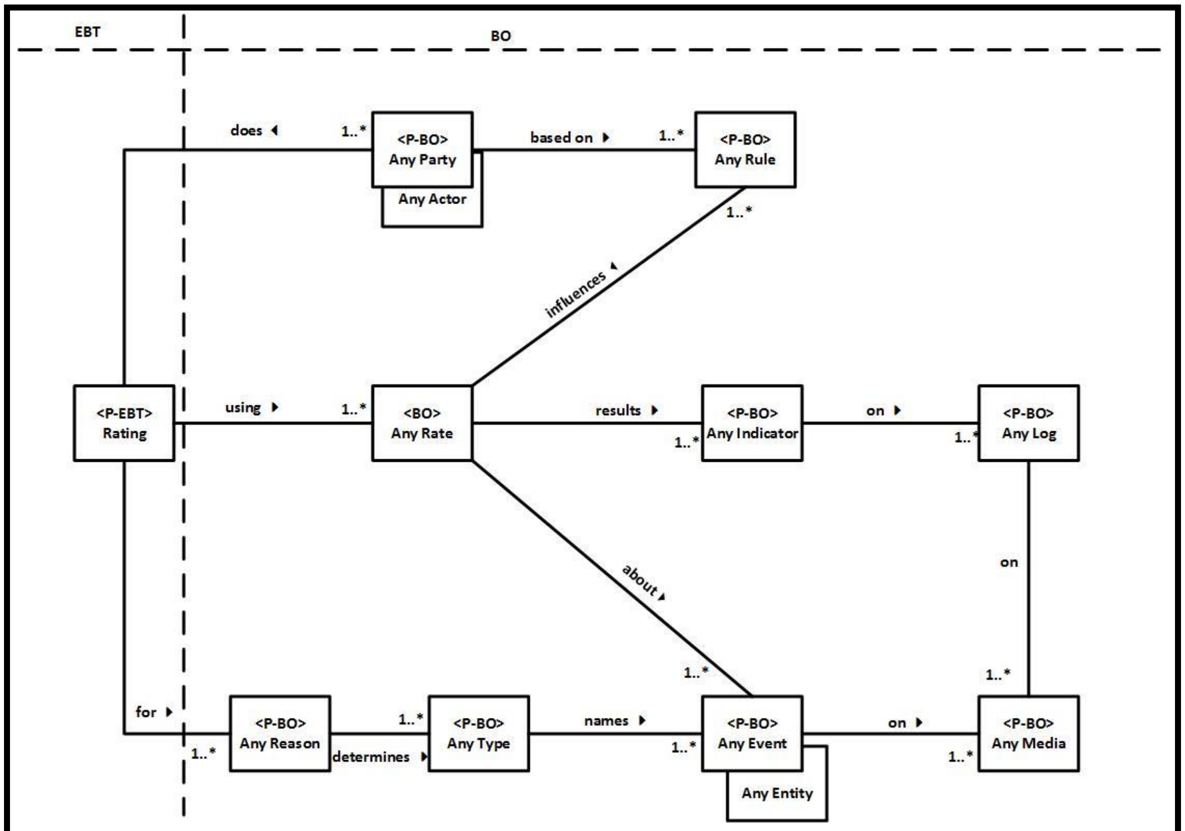


Figure B2.5: AnyRate SDP

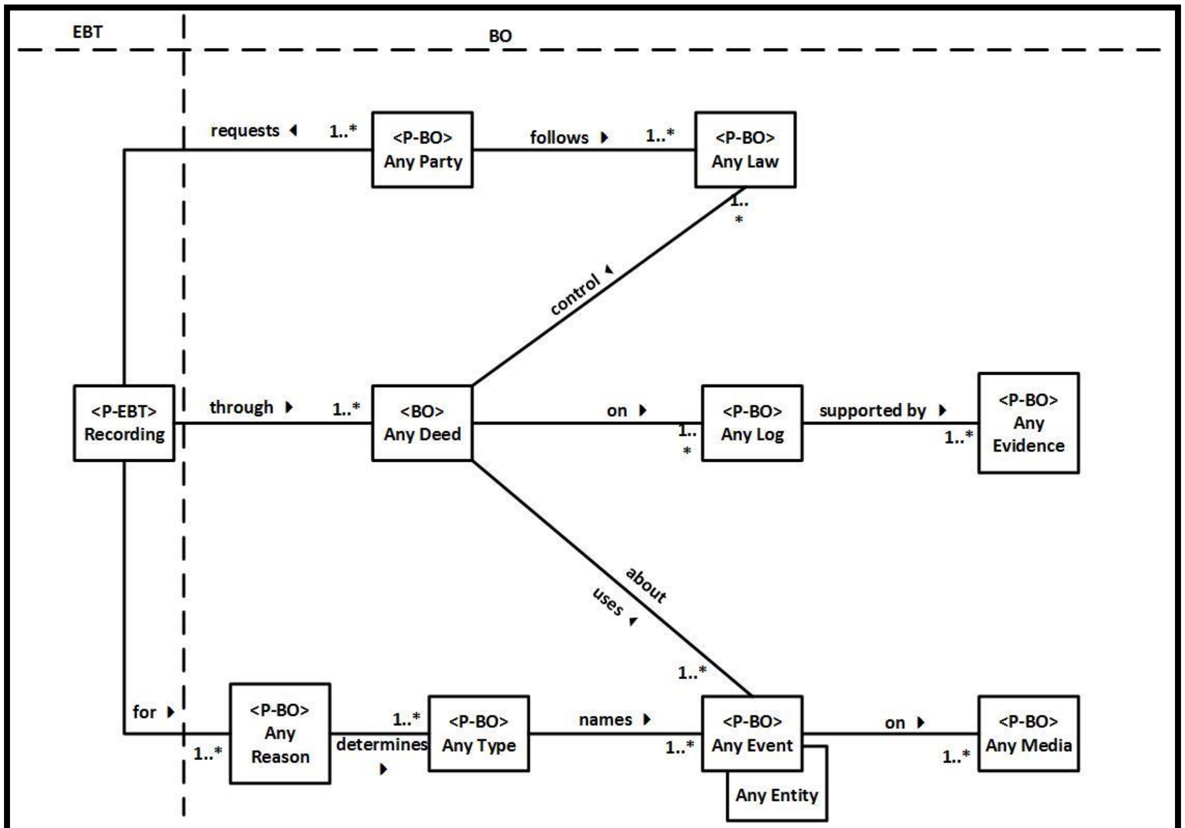


Figure B2.6: AnyDeed SDP

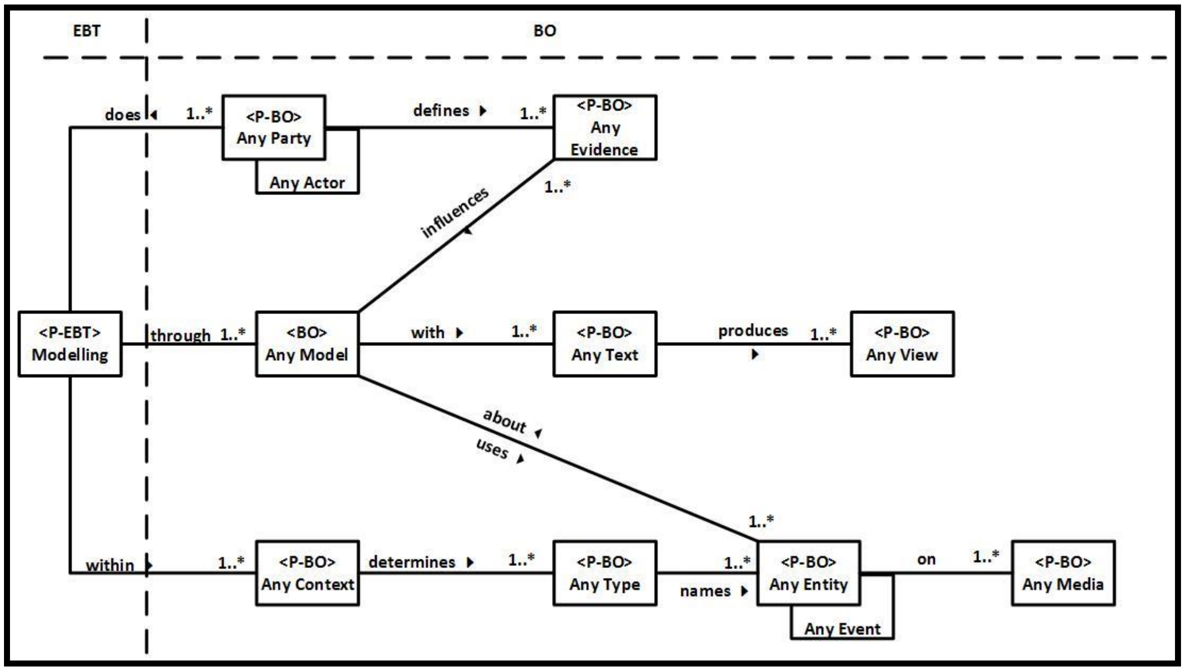


Figure B2.7 AnyModel SDP

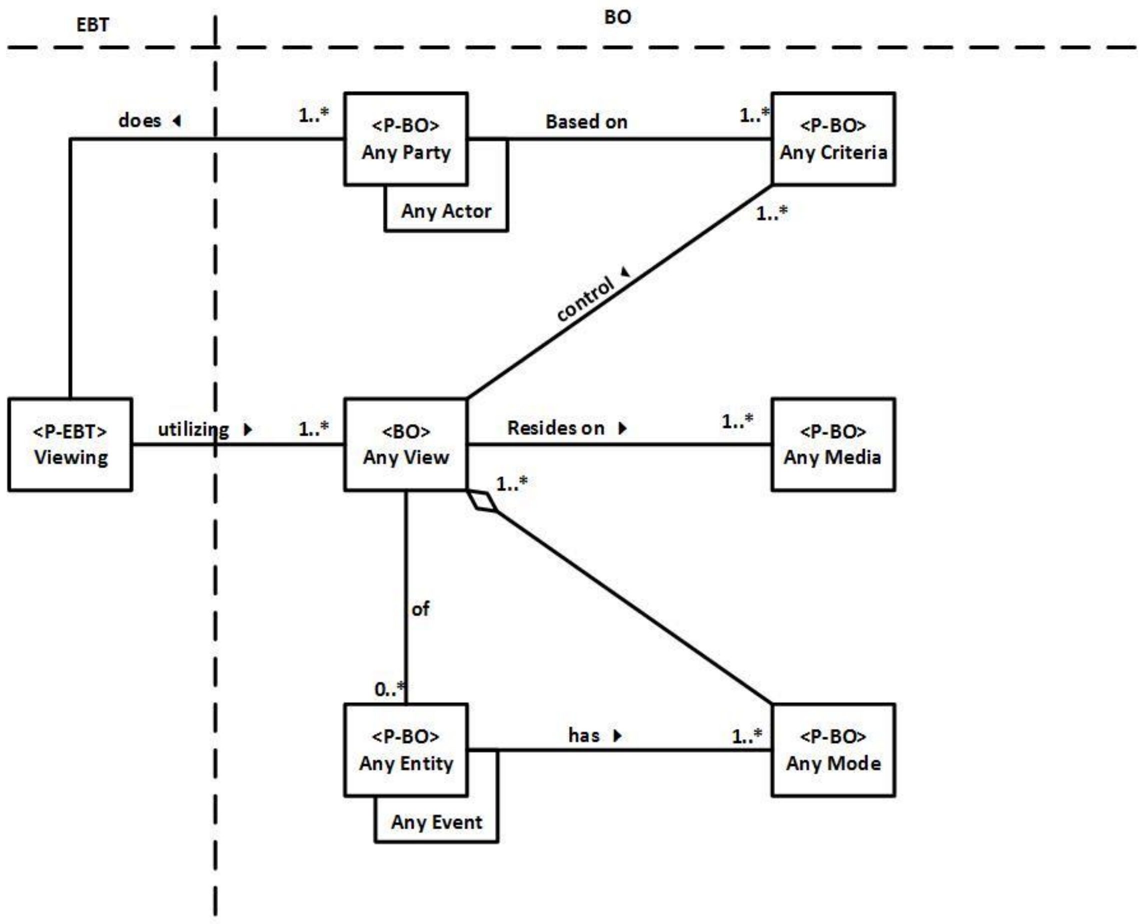


Figure B2.8: AnyView SDP

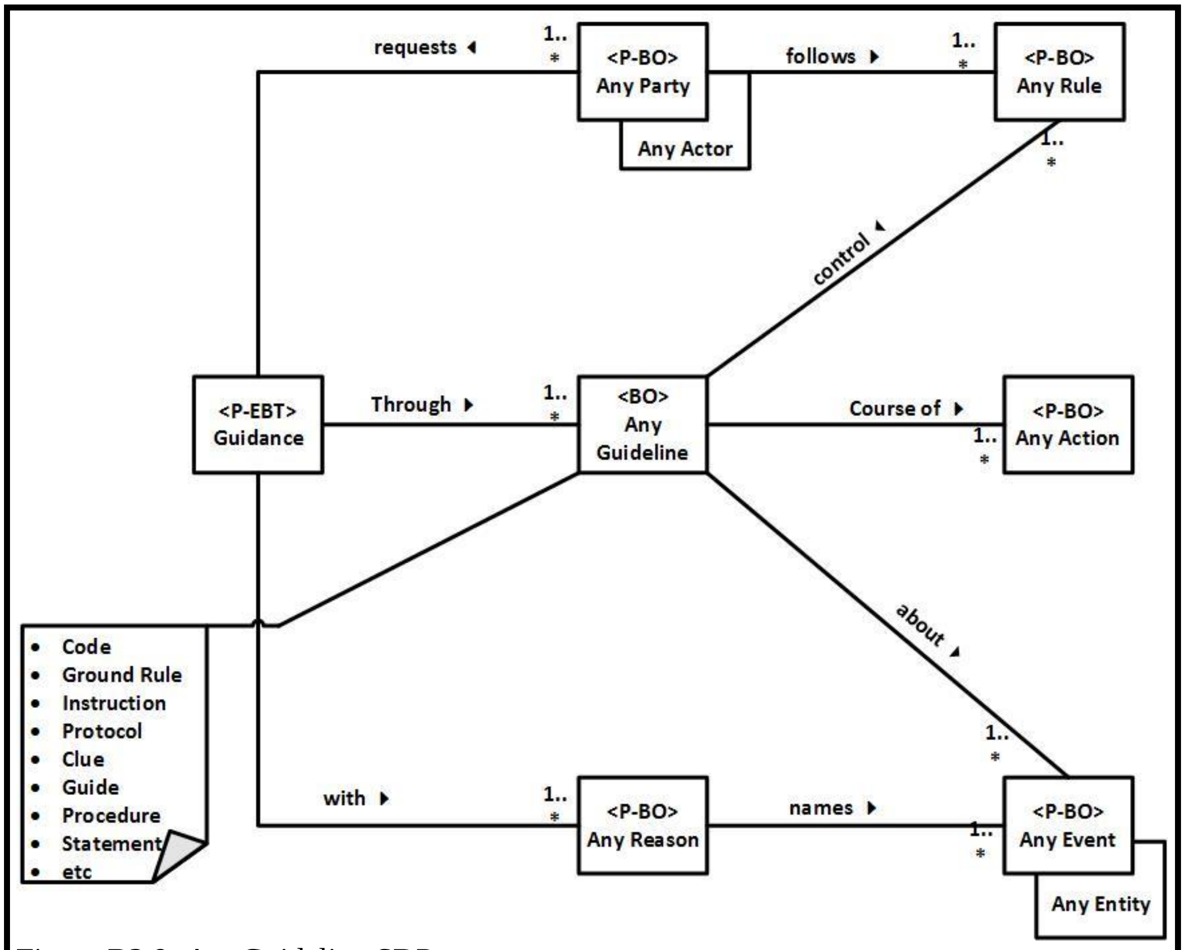


Figure B2.9: AnyGuideline SDP

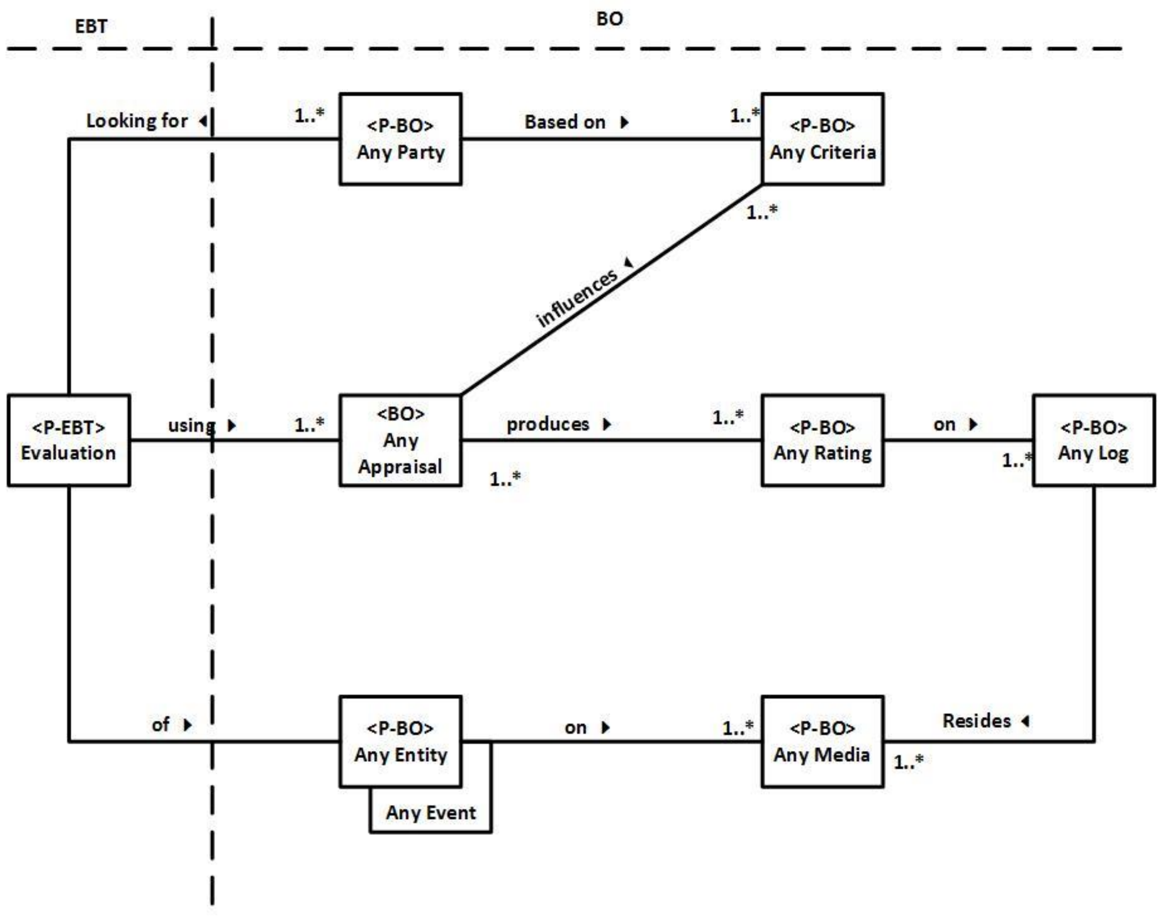


Figure B2.10: AnyAppraisal SDP

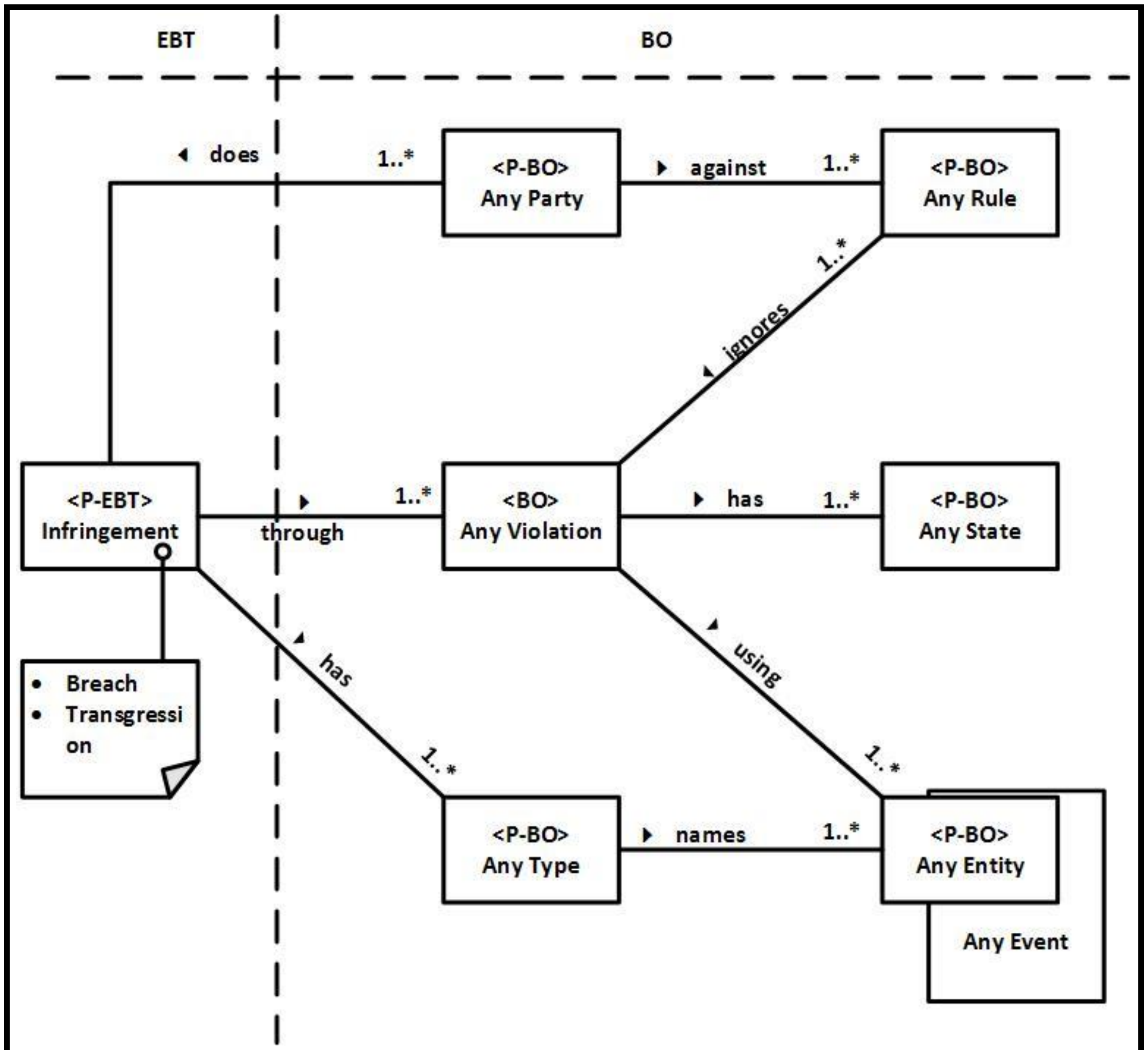


Figure B2.11: AnyViolation SDP