

The AnyCorrectiveAction Stable Design Pattern

Shivanshu K. Singh¹ and Mohamed E. Fayad²

Department of Computer Engineering,
Charles W. Davidson College of Engineering,
San Jose State University, San Jose, CA, USA.
¹*shivanshukumar@gmail.com*, ²*mefayad@gmail.com*

Abstract

The stable design pattern AnyCorrectiveAction is based on the term Corrective Action [1], which generally refers to any counter measure taken by any party, who seeks Maintenance by the solution to any problem. Maintenance can be of different types: Preventive, Adaptive, Predictive and Corrective. Any Party decides upon taking any Corrective action, based on any evidence and the reason, and also the criteria that govern it. It involves one or more counter measures taken to prevent the recurrence of the problem or the reason for the corrective action. At the end of it, a log is produced, which serves as a basis for verification of the fact that the corrective action has been accomplished and it meets the ultimate goal of maintenance. This paper aims at applying the Software Stability Model [2] approach towards modeling a system that has to perform maintenance, by employing a Corrective Action and come up with a stable pattern, which can form a part of such a system, focusing at applying corrective action in a holistic way, rather than tying it on only one application specific context, which might cause potential impedance mismatch between process and workflows at a later stage. The Software Stability Model ensures high reusability, ensuring that a design once created can be used to model any application, in any domain, thus making the task of designing more efficient, and the modeled system, largely domain independent [3]. The goal of this paper is to design a Stable Pattern for Corrective Action and to model the system on the basis on an Enduring Business Theme. Here, the ultimate goal or the EBT is Maintenance and AnyCorrective action is what acts as a workhorse to achieve the goal of maintenance. In this paper, we have first designed a model, which defines the relationship between the Enduring Business Theme through Business Objects. We have then gone ahead with modeling an application scenario, which portrays the reusability of the developed stable pattern. A comparison of the Traditional Model and the Stable Model has also been included to describe how the latter overcomes to drawbacks of the former. Different models, such as, use cases, CRC cards, class diagrams and sequence diagrams have been used to give a better insight into the pattern and possible applications of it.

1. Introduction

The Traditional Model of software modeling does not take into consideration issues such as high reuse stability, scalability, impedance mismatch etc., and systems as well as patterns designed on those lines fail to be generic enough that they can be applied in any scenario regardless of the domain of application. Any software designed on traditional lines lacks the flexibility of being adapted to any other application. Any Software modeling performed by using that technique is never flexible and it will not be adaptable to changes. It's a static view of the system, and also lacks the ability to depict the dynamic nature of the system.

Also, in the software development industry, numerous projects have failed and they continue to do so. Adding to this, the ones which do not fail have huge costs of maintenance associated

with them. This is primarily because there is a flaw in the analysis and design of those systems and the way they have been modeled. They have been modeled keeping one static configuration or only one application in mind and they lack the ability to dynamically adapt to changes. It is like constructing a big car manufacturing plant that produces just one car out of the facility and then re-structuring the whole plant again to produce the second car and so on, which clearly is an extremely inefficient way of doing anything.

So, now we are trying to model on the lines of making a system dynamic and instead of concentrating on a particular application as such, we are now concentrating upon a Stable Design Pattern of Corrective Action and the Enduring Business Theme or EBT that it plans at implementing. The EBTs are realized through their workhorses, known as Business Objects or BOs, which are then applied to one or more applications. We call this as the Stability modeling technique for software modeling. It gives us the ability to cater to various Business Themes and make the system domain independent rather than sticking to just one particular application of it. We hereby attempt to show how the same system, created earlier by using traditional model can be modeled by using the stability model, thus giving it a larger purview and stability over time.

2. Pattern Documentation

- **Pattern Name:** AnyCorrectiveAction Stable Design Pattern

The stable design pattern AnyCorrectiveAction is based on the term Corrective Action, which generally refers to any counter measure taken by any party, who seeks Maintenance by the solution to any problem. Maintenance can be of different types: Corrective, Adaptive, Predictive and Corrective. Any Party decides upon taking any Corrective Action, based on any evidence and the reason and also the criteria that govern it. It involves one or more counter measures taken to prevent the recurrence of the problem or the reason for the corrective action. At the end of it a log is produced, which serves as a basis for verification of the fact that the corrective action has been accomplished and it meets the ultimate goal of maintenance. AnyCorrectiveAction is a Business Object to a workhorse to realize the Enduring Business Theme of Maintenance.

- **Known As:**

Betterment

The AnyCorrectiveAction pattern is sometimes also confused with the term betterment, i.e. making something that is there and making it better, but being confused as Corrective Action, but it's not quite the same, since betterment is about making something that is there, which is already in good shape and improving upon it. This is not what a corrective action does. Our pattern is employed, where we want to rectify a problem that exists, instead of making something which may be good, even better.

Prevention

Prevention is again, not about taking a corrective action. It is rather about taking preventive measures that help the system to avoid the occurrences of any new problems that would need a corrective action in the future. It is, hence, not the appropriate term for our pattern.

Correction

Correction, in some cases, can also mean the comments given for what should have been the right thing, instead what is currently there. [4] Here, we are not talking about it and since Correction encapsulates all such possibilities, i.e. the responses to any problem or way of doing things that currently exists, it is not accurate enough to be fit into the definition of the pattern in question. The term correction can also be the measure of changes made to anything to achieve the desired level/state, it might even be considered sometimes as a punishment for some wrong deeds. [5] Therefore, because the term ‘correction’ can have multiple meanings under different contexts, it is not accurate enough and it can’t be used as the term which would represent the pattern AnyCorrectiveAction.

- **Context:**

The Stable Design Pattern, CorrectiveAction can be applied to any domain in which we have a board, which requires maintenance and which has a requirement of doing so, because of a problem that exists. For example, in military, there can be a corrective action taken, like initiating a court marshal on an officer, if the problem of smuggling of arms was the case and there was evidence that supported the act of smuggling. Generally applied in military domains, this pattern can now be applied across domains and by the means of Stability model, it has now attained a completely domain independent form. It can be applied to any organization i.e. a company that wants to raise working capital can make a corrective action of liquidating unused assets, which were bought earlier, but are not being used, or this can be applied to any university, where the university takes a corrective action of counseling students for not cheating, as a corrective action to maintain academic honesty or to counter the problem of plagiarism.

3. Problem

The pattern AnyCorrectiveAction must incorporate the way corrective action is taken by AnyParty to overcome any recurring problem. There has to be a valid reason that prompts maintenance of the system for which the corrective action is required. The corrective action must be approved by an authority or AnyParty, and has to fall under the periphery defined by the policies or AnyCriteria set for by the authority. Finally, AnyCorrectiveAction must result into AnyEvidence that shows that the corrective action was taken and should also be supported by AnyLog. The approving authority has to be displayed the Evidence and Log to prove that the corrective action was already taken.

3.1. Functional Requirements

Maintenance:

Maintenance should correctly restore back the right state of the system. Maintenance would be achieved by AnyMechanism that is employed for this, but it’s the job of maintenance as such, to make sure that the system now acts in the correct way that it should have after the corrective action was taken.

AnyParty or AnyActor:

AnyParty or AnyActor specifies AnyCriteria that governs the mechanism being used and the measures being taken in order to perform the corrective action. The one or more parties and persons must have someone with them, who has the expertise related to the domain in which the corrective action has to be done, so that the required corrective action(s) can be aptly identified, prioritized and instituted for action. Also, the party here is the authority, who establishes the fact that the corrective action was properly carried out by examining the logs

and evidence generated. So, the expert(s) is (are) needed, so that it can be judged that whether the corrective action was properly applied or not.

Issuing Authority:

There has to be some governing body that seeks maintenance to a problem and approves the corrective action. It cannot be done in an *ad hoc* manner by just anyone.

Criteria:

AnyCorrectiveAction must be governed by the Criteria set forth by the issuing authority and must be performed under the boundary of one or more Criteria that have been set. It cannot go beyond the policies that drive the business; otherwise, the corrective action would result in other problems for which more corrective actions would be required.

Moreover, AnyCriteria goes beyond just what AnyParty or AnyActor has defined with respect to themselves and their internal concerns; certain governing criteria can also come from the law of the land and the culture of the society that the system should cater to. Such criteria have to be met.

Criteria and Corrective Action definition process:

Defining AnyCriteria is also one point of concern especially, when multiple AnyParty and AnyActor are involved in the formation of one or more AnyCriteria. There has to be an effective coordination mechanism and a formal process of doing this, because the definition of criteria has to be carried out responsibly and effectively. Each participating member must review and approve the decisions being taken and for this, a formal process needs to be instituted.

AnyMechanism:

Some Mechanism has to be used in order to perform CorrectiveAction and carry out Maintenance, so such a Mechanism must exist. AnyMechanism would be governed by the Criteria defined by AnyParty or AnyActor seeking Maintenance. It is this Mechanism that achieves Maintenance. AnyCorrectiveAction in this case can be a type of AnyMechanism.

Type of Mechanism used:

The type of mechanism used to attempt maintenance through AnyCorrectiveAction has to be specific to the problem, which has to be addressed and it should be within context. Choosing the right mechanism is essential to the Corrective Action's appropriateness.

Proof:

The corrective action must lead to an evidence or proof that shows that the corrective action was taken, otherwise, the purpose of corrective action is defeated and it cannot be made sure that whether the corrective action was taken in an appropriate manner or not.

Collection of data:

There should be provisions for capturing the related data, which results from the effects of the corrective action and must be captured in an AnyLog, which can display the chronological order of the various actions and effects of this corrective action.

Reason:

There has to be a well defined and identified reason for the corrective action to be performed and hence the maintenance to be undertaken. Generally for employing Corrective Action, the problem at hand is a recurring one. Thus, the reason has to be a recurring problem that occurs

within the system. It has to be something internal to the system and external influences cannot be considered as reasons that require any corrective action on the system. One or more AnyReason defines the type of Mechanism to be used to achieve Maintenance, depending upon what the reason is and the various factors and needs associated with them.

Type: Maintenance can be of different types: Corrective, Adaptive, Predictive and Corrective. Each one of these types will lead to special type of entities that needs to be included in the corrective action process. It is essential that such ‘type’ of Maintenance is clearly identified or a sub system is out in place that identifies the right kind of Maintenance that has to be done.

AnyEntity:

AnyEntity is something that is used by AnyMechanism that is being employed, to perform Corrective Action and do Maintenance. So, AnyEntity being used by AnyMechanism has to be accessible and interface-able with the system, so that it can be used in performing the required tasks. Choosing the right entity is crucial here, since it may be governed by certain access regulations and there may be some other limitations to the use of it, so to maintain the cost effectiveness the entity must be chosen correctly. The Entity must therefore make the relevant meta data available upon request, so that the other parts of the system can properly decide, if it can be used or not and if so, can properly interface with it to get the job done.

AnyLog:

AnyLog must have at least 3 files: the log of the problem that exists, the corrective action process to be executed, and the execution log generated from the corrective action process. AnyLog must be able to convey the related data to the stakeholders and if required, the system should be able to interface with the medium that any of the stakeholders wants to have the log on.

3.2. Non Functional Requirements

Urgency:

At any time, there may be a certain set of problems that need to be addressed and not just one such problem alone. In such a scenario, one problem may be more severe than the other and might require urgent attention. The system must also take into account such situations, where a bunch of corrective actions are required to be taken, but we need to prioritize them based on the urgency of the same. This may be decided upon by the issuing party as to which one to do first and which one the next.

Cost effectiveness:

The corrective action being taken must be done in a cost effective manner. The system must not take forever to perform the tasks that are to be performed as a part of the process and still the entire process should be limited within a set amount of budget, so as to keep a balance between performance and costs incurred. Any extremes that occur here have the tendency of stalling the system, if it takes too long to finish the job or making it impossible for the management to keep the system in use, if it bears an enormous cost.

Exactness:

The corrective action taken should exactly conform to its description and must act fully and exactly as per the boundaries specified for it. Any more or less than the expected action, would render it useless and may even result in a situation, where one or more corrective actions are required to counter the ill-effects that it created in the processes of attempting the

corrective action that was instituted. If the corrective action is not exact to its definition, more corrective actions would be required and that would result in increased cost, time and effort.

Reliability:

The reliability of any corrective action is also desirable, because, if one is not sure about the outcome of the corrective action being taken, it cannot be predictable and hence would not be as effective in addressing the problem as it should be, thereby defeating the whole purpose of serving it only partially.

Also, the reliability and consistency of AnyLog and AnyEvidence is required, as if they are not reliable enough, there will not be a proof to the fact that the corrective action was taken and its effectiveness cannot be measured.

3.3. Challenges and Constraints

3.3.1. Challenges

1. Maintenance and Corrective Action as such, needs to be performed under certain constraints and by keeping in mind certain criteria. However, to come up with such criteria and to determine the right set of criteria is a challenge, while doing maintenance on the system.
2. There can be many criteria based on which, the Maintenance has to be performed, for any system. There may be some of them, which contradict each other; this is a challenge while defining the criteria.
3. Corrective Action can be applied in varied ways. Maintaining a system through Corrective Action would need certain mechanism to be used. Choosing the right mechanism that correctly enables performing the Corrective Action, is a challenge.
4. Accuracy is a big challenge, because there can be a corrective action for a problem, but things like learning and a learning plan for example, when used as corrective action, are heavily dependent upon learning abilities of the participants and so the corrective action though appropriate, but may not eventually result in a desired outcome and other measures might also be required. Hence, it is a challenge to make AnyCorrectiveAction highly accurate.
5. There must be a proper reason for which maintenance is required and that too one that is of corrective in nature such that AnyCorrectiveAction needs to be employed. Sometimes, some thing close to the actual reason of a recurring problem can be confused to be the real cause of the problem that exists, because of which the system requires maintenance. In such a scenario, the efforts would be made to address this secondary reason, or the symptom, while the main reason would by mistake, get neglected and then even though the corrective action would be taken, it would fail to address the real reason, thereby rendering the whole exercise fruitless.
6. Recording the process of Maintenance and Corrective Action is a challenge in itself and the process, as well as the various factors considered in the process, findings etc should be recorded to make sure that the problem that caused the need for Maintenance is not repeated in the future.

The entities being used in the system for any purpose can pose a challenge for the system. If there are certain entities governed by access restrictions or usage policies, then to conform to these can be a big challenge. To add to it, when you consider many such entities being governed by different governing bodies/rules, it multiplies the complexity of the system. Hence, it has to be much more efficient in managing these entities and performing tasks like workload distribution etc., more the number of such requirements, the bigger the challenge would be.

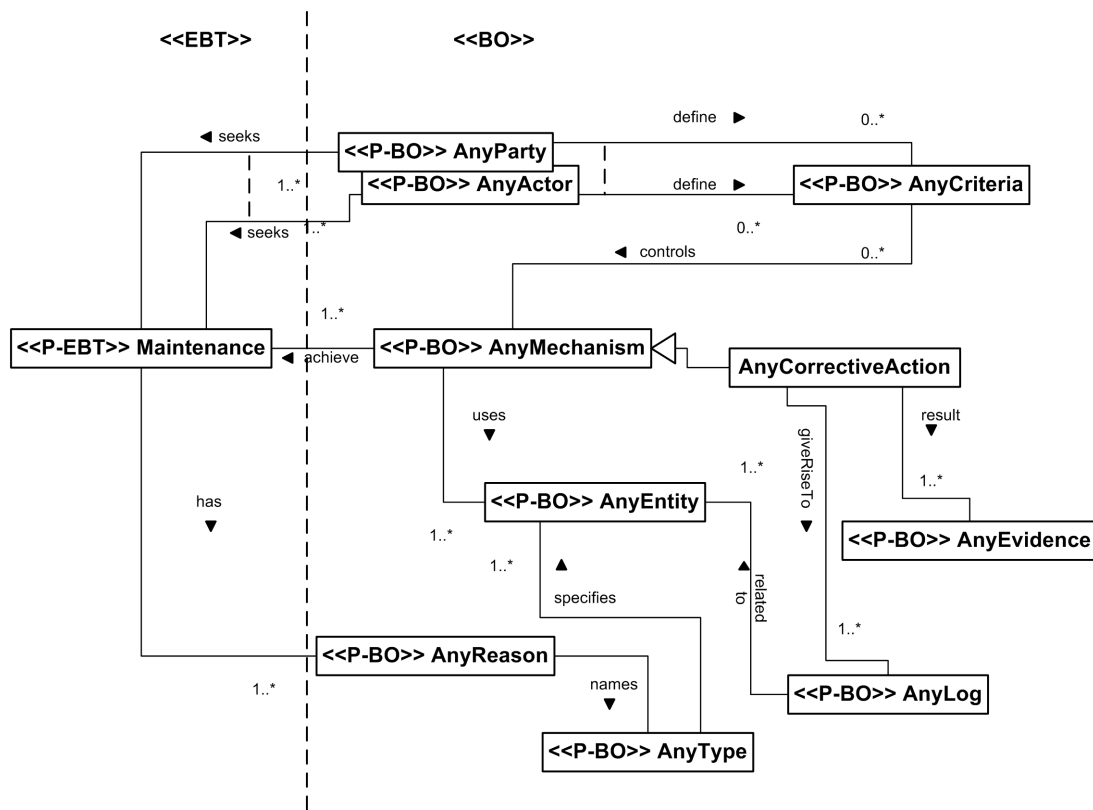
7. Urgency of any corrective action may require the action to be taken within specific time and to meet stringent deadlines is a challenge in itself, on top of this, if there are more than one extremely high priority tasks to be done, then this efficiency has to be maintained at its best, else the system would fail to serve its purpose. Convoluting this with cost effectiveness raises even bigger challenges.

3.3.2. Constraints

1. The Maintenance has to be performed within a certain set of criteria, depending upon the context in which the Maintenance is being performed.
2. Maintenance through Corrective Action can be performed only on a problem that is recurring and something which otherwise is not solvable.
3. A governing authority should approve Corrective Action before it is put into action. The Maintenance process that follows must document the procedure and log the results so that the execution of it can be verified.
4. Corrective Action can never go beyond the purview of the laws of the land and all the actions mandated as a part of that maintenance must conform to the law.

4. Solution

4.1. Pattern Structure



- * <<P-EBT>> - Pattern EBT
- * <<P-BO>> - related Pattern BO, which, along with AnyCorrectiveAction and other pattern BOs, realize the EBT of Maintenance
- * BO = Business Object
- * EBT = Enduring Business Theme

Figure 1 - Pattern Class Diagram

Class Diagram Description

1. AnyParty or AnyActor seeks Maintenance, which is achieved through AnyMechanism
2. AnyParty or AnyActor defines AnyCriteria, which Controls AnyMechanism
3. Maintenance has AnyReason, which names AnyType, which specifies AnyEntity used by AnyMechanism
4. AnyCorrectiveAction results into AnyEvidence and gives rise to AnyLog

4.2. Participants

Classes:

AnyCorrectiveAction: any thought that is conceived by any actor.

Patterns:

Maintenance:	represents the enduring concept of Maintenance, includes the operations and attributes of the same.
AnyEntity:	refers to any objects that are being used in the system.
AnyActor:	person, animal, software or hardware; something that acts on its own will.
AnyParty:	any person, organization, country or political party.
AnyEvidence:	something that forms the basis for AnyCorrectiveAction
AnyCriteria:	refers to the basis that governs AnyMechanism through which Maintenance is achieved
AnyType:	is the type of AnyMechanism and it specifies AnyEntity that shall be used by AnyMechanism
AnyMechanism:	the method employed to achieve Maintenance
AnyLog:	the record that is generated after AnyCorrectiveAction is taken
AnyReason:	the problem that exists and which requires corrective action.

4.3. CRC Cards

The CRC cards are designed based on the guidelines as mentioned in [6, 7]. They help us in defining a complete picture of how a common set of operations, with individual responsibility of each class and the related operations served by it, can be utilized to form the basis of an application scenario. Such an application that is built on top of this pattern, utilizing one or more operations from the generic operations that are offered by the BOs, has been shown in subsequent section 5 (Applicability). The CRC-cards name the class, responsibility, and its collaborations. The CRC-cards also name a role for each class, which is useful for identifying the class responsibility. Each class should have only one and unique responsibility. The collaboration consists of two parts: clients and server. Clients are classes that collaborate and have relationship with the named class. Server contains all the services that are provided by the named class to its own clients. It is worth to point out that in documenting CRC-cards for stable patterns, we deal with any pattern that are included within the main pattern itself as a class. That is, each sub-pattern will be represented by a CRC-card that documents its responsibility and collaborations as a black box. To avoid any confusion, and for simplicity, we do not care about how the sub-pattern handle its reasonability according to its internal structure, all what we care about here is that this sub-pattern will perform the task as a black box, leaving the other details to the second abstraction level of the pattern description.

<<P-EBT>> Maintenance(Maintenance)		
Responsibility	Collaboration	
	Clients	Server
To achieve consistency	AnyParty AnyActor	maintain() renew() increaseLifeSpan()
Attributes:		
type, duration, degree, frequency, phase		

<<P-BO>> AnyEntity (AnyEntity)		
Responsibility	Collaboration	
	Clients	Server
to help in accomplishing a task	AnyMechanism AnyType	describe() modify() helpAccomplishTask()
Attributes:		
name, domain, state, size, description		

<<P-BO>> AnyCorrectiveAction(AnyCorrectiveAction)		
Responsibility	Collaboration	
	Clients	Server
To take counter measure and produce evidence of it	AnyMechanism AnyEvidence AnyLog	takeCorrectiveMeasure() restore() produceLog() resultEvidence()
Attributes:		
duration, name, type, serialNumber, purpose		

<<P-BO>> AnyActor (AnyActor)		
Responsibility	Collaboration	
	Clients	Server
to think and take action	Maintenance AnyCriteria	takeAction() defineCriteria() seekMaintenance()
Attributes:		
name, address, occupation, idNumber, dateOfBirth		

<<P-BO>> AnyType (AnyType)		
Responsibility	Collaboration	
	Clients	Server
specify the entity to be used	AnyReason AnyEntity	specifyEntity() signify() determine()

Attributes:
name, usage, classification, structure, value

<<P-BO>> AnyReason (AnyReason)		
Responsibility	Collaboration	
	Clients	Server
to serve as the cause for which maintenance is required	Maintenance AnyType	nameType() reflect() corroborate()
Attributes:		
context, validity, significance, stimulus, name		

<<P-BO>> AnyCriteria (AnyCriteria)		
Responsibility	Collaboration	
	Clients	Server
to constrain any action being taken	AnyParty AnyActor AnyMechnism	control() restrict() formBasis()
Attributes:		
domain, type, clauseNumber, name, effect		

<<P-BO>> AnyEvidence(AnyEvidence)		
Responsibility	Collaboration	
	Clients	Server
serve as a proof	AnyCorrectiveAction	prove() justify() assert()
Attributes:		
proof, timestamp, type, domain, assertion, reliability		

<<P-BO>> AnyParty(AnyParty)		
Responsibility	Collaboration	
	Clients	Server
To manage	Maintenance AnyCriteria	takeAction() defineCriteria() seekMaintenance()
Attributes:		
partyName, purpose, motto, type, location, size		

<<P-BO>> AnyLog (AnyLog)		
Responsibility	Collaboration	

To list a chronological sequence	Clients	Server
	AnyEntity AnyCorrectiveAction	document() record() storeInfo()
Attributes:		
name, size, type, media, logNumber, timestamp		

<<P-BO>> AnyMechanism (AnyMechanism)		
Responsibility	Collaboration	
to attempt doing maintenance	Clients	Server
	Maintenance AnyCriteria	operate() design() conduct() achieve()
Attributes:		
reason, name, effectiveness, strength, type		

4.4. Consequences

The AnyCorrectiveAction Stable Design Pattern presents us with many challenges that have to be tackled. In order to meet those challenges and to fulfill them, we would need to design the system taking into consideration, certain capabilities that the system must have and doing so might result in many consequences that the system will face. Here are a few of those:

1. To be able to support a large amount of media for storing AnyLog generated by the system, it would need to have the capabilities for interfacing with the different types of media that are available. This would result in the system being of a large size, a considerable part of which would be something, which deals with this interfacing only, than anything else.
2. Also, if the user is to define the way to handle the storage, then we need to have the support for different storage mechanisms. If these storage facilities are on the internet and need an access mechanism, then the system needs to cater to that too, and things like access rights management and other issues also need to be implemented.
3. These features may also call for support for plug-ins and an appropriate API would be needed, so that the consumer can make use of the available options within the system, as well as add his/her custom devices, by integrating the necessary plug-ins into the system.
4. Doing so, we may also have to consider many things like security, because we would allow the system to be customized by the client to a great extent and hence whatever the plug-in does need to be monitored or approved before it can be added, or alternatively such things have to be 'legally' taken care of and further investigation needs to be done in this area.

- For the system to be cost effective and to deal with the issues like critical corrective actions, especially in those cases, where more than one corrective actions are of a very high priority and it is necessary to handle both of them as quickly as possible, then we need to empower the system with efficient scheduling capabilities, where the system can effectively decide which entity to use and at what time and for how long. The system needs to plan out for minimum access of the entities, if they are to be paid for on a pro-rata basis and hence the system should be able to arrange the set of activities, on the fly, in a way which strikes a balance between cost and performance, while staying within the cost and time constraints.

5. Applicability

5.1. Application 1: Company trying to reduce losses

This application is about a company, which is undergoing losses, because some of its employees lack minimum attendance. So, in order to reduce these losses incurred because of low attendance, the company takes corrective action in that regard.

5.1.1. Class Diagram

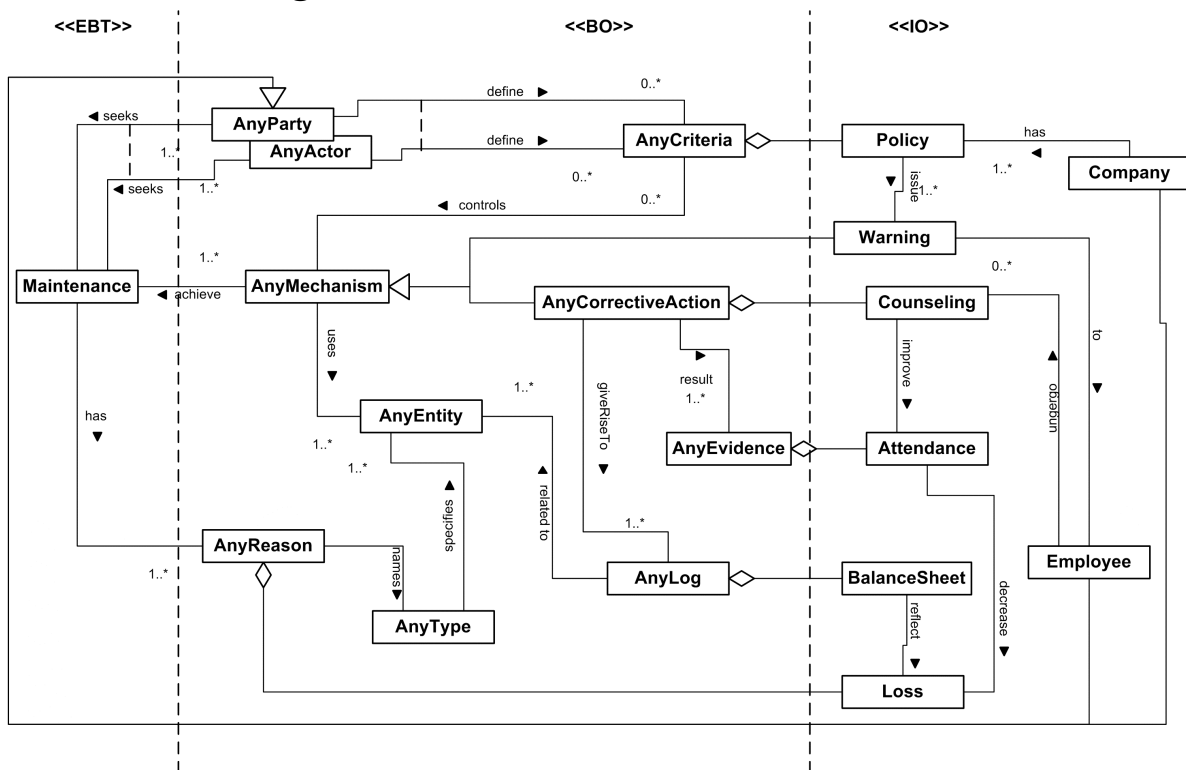


Figure 2 - Application 1 Class Diagram

5.1.2. Class Diagram Description

- AnyParty or AnyActor seeks Maintenance, which is achieved through AnyMechanism
- AnyParty or AnyActor defines AnyCriteria, which Controls AnyMechanism

3. Maintenance has AnyReason, which names AnyType, which specifies AnyEntity used by AnyMechanism
4. AnyCorrectiveAction results into AnyEvidence and gives rise to AnyLog
5. Company has a Policy, which issues one or more Warning to Employee
6. Employee undergoes Counseling, which improves Attendance
7. Attendance decreases Loss
8. BalanceSheet reflects the Loss

5.1.3. Use Case

Title	Reduce Losses Due To Improper Attendance		
Id	1.1		
Actor	Roles		
AnyParty	Company, Employee		
Class	Type	Attribute	Operation/Interface
Maintenance	EBT	duration, degree	renew()
AnyReason	BO	validity, significance, stimulus	corroborate()
AnyParty	BO	partyName, motto, type, size	defineCriteria()
AnyMechanism	BO	name, effectiveness, strength	achieve()
AnyEntity	BO	description, name	helpAccomplishTask()
AnyCriteria	BO	clauseNumber, type, domain, name	control()
AnyCorrectiveAction	BO	duration, name, type, serialNumber, purpose	produceLog() resultEvidence()
AnyEvidence	BO	timestamp, assertion, reliability	prove()
AnyLog	BO	name, size,	record()

		media, logNumber, timestamp	
Loss	IO	cost, type	affectNegatively()
Company	IO	name location objective	seekMaintenance()
Policy	IO	text, clause, name	institureWarning()
Counseling	IO	duration, reason, domain	motivateEmployee() improveMentality()
Attendance	IO	registerName, size	showPresense() keepTrack()
BalanceSheet	IO	name, day, month, year	elicitateFinance() storeRecord()
Employee	IO	id, name	undergoCounselling()
Warning	IO	type, instruction	inform() issueWarning()

Description

1. AnyParty seeks Maintenance

1.1 which has AnyReason

1.2 AnyReason determines AnyType

TC: Maintenance is done by what? Who seeks Maintenance? Maintenance is based on what? Why is Maintenance required? What determines AnyType? How?

2. Maintenance

2.1 is achieved through AnyMechanism,

2.2 A Company seeks Maintenance, because of the problem of Loss and

2.3 uses Warning to intimate to the Employee about it as it is being created, because of Employee's low Attendance

TC: what achieves Maintenance? How is Maintenance achieved? Why is Maintenance needed?

3. AnyType

3.1 specifies AnyEntity which

3.2 is used by AnyMechanism

3.3 to achieve Maintenance

TC: AnyType does what? How? What specifies AnyEntity? Based on what? What does AnyMechanism use to achieve Maintenance?

<p>4. AnyParty</p> <p>4.1 defines AnyCriteria</p> <p>4.2 Company defines a Policy which</p> <p>4.3 Governs the code of conduct of Employees</p> <p>4.4 And describes the process of issuing Warning to Employees</p> <p>TC: who defines AnyCriteria? AnyParty does what? Based on?</p>
<p>5. AnyCriteria</p> <p>5.1 controls AnyMechanism</p> <p>5.2 issues Warning to the Employee</p> <p>5.3 uses Warning to intimate to the Employee about low Attendance, since it is creating Loss</p> <p>TC: what does AnyCriteria do? What controls AnyMechanism? Is this control needed? How? Who decides on AnyCriteria?</p>
<p>6. AnyCorrectiveAction</p> <p>6.1 results into AnyEvidence</p> <p>6.2 and gives rise to AnyLog</p> <p>6.3 the Employee undergoes Counseling which</p> <p>6.4 helps him/her get perspective and understand the expectations of the Company along with undergoing introspection</p> <p>6.5 and Attendance is increased as a result</p> <p>TC: what does AnyCorrectiveAction result? What else does it give rise to? What produces AnyEvidence? Or AnyLog? What do they do?</p>
<p>7. AnyLog is</p> <p>7.1 related to AnyMechanism</p> <p>7.2 which achieves Maintenance</p> <p>TC: what is related to AnyMechanism? How? AnyLog is related to what? What achieves Maintenance?</p>
<p>8. AnyLog</p> <p>8.1 helps prove that any AnyCorrectiveAction was taken</p> <p>8.2 undergoing Counseling helps the Employee to</p> <p>8.3 improve Attendance</p> <p>8.4 which reduces Loss</p> <p>8.5 which is shown by the BalanceSheet</p> <p>TC: what gives a proof of AnyCorrectiveAction? What proves that AnyCorrectiveAction was taken? What else does this do? How?</p>
<p>Alternatives</p>
<p>1. Company seeks Maintenance to the problem of Loss</p> <p>2. which is being generated due to lack of Attendance</p> <p>3. Company institutes a policy for incentives for better attendance</p> <p>4. and raises the salaries</p> <p>5. which motivates the Employee to have a better attendance record</p> <p>6. which helps in reducing the Loss to the Company.</p>

5.1.4. Behavior Diagram

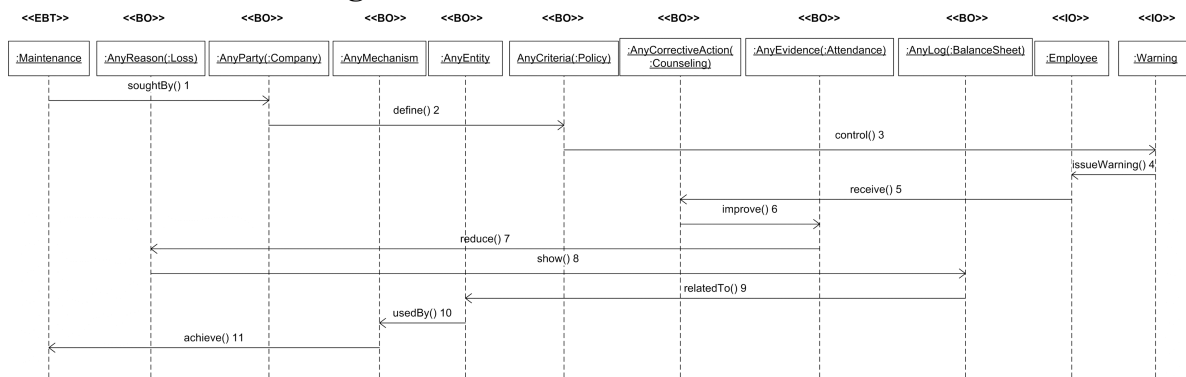


Figure 3 - Application 1 Sequence Diagram

6. Related Pattern and Measurability

6.1. Related Pattern: Traditional Model

AnyCorrectiveAction Stable Design Pattern has the goal of Maintenance. However, this is something that can be confused with e.g. Betterment, Prevention, Correction etc., as we saw earlier in the ‘Known As’ section of the pattern documentation. Moreover, there are many ways that Maintenance through Corrective Action has been attempted and is being attempted. There exists however, no such stable pattern that captures the essence of achieving Maintenance though employing Corrective Action in a way that is holistic and stable. Due to lack of such a stable pattern, this section illustrates a non – stable pattern approach to doing corrective action. Since, there exists no stable pattern on Corrective Action, presented here is a ‘traditional’ approach (**Figure 4 - Traditional Model for the pattern**) to the same, along with a comparison (Table 1 - Comparison between Traditional Model (TM) and Stable Pattern (SP) based on Modeling with the Stable Pattern based approach, on the basis of how well does each of the methods satisfy one or more Modeling Adequacies towards building an application that employs Corrective Action and does Maintenance.

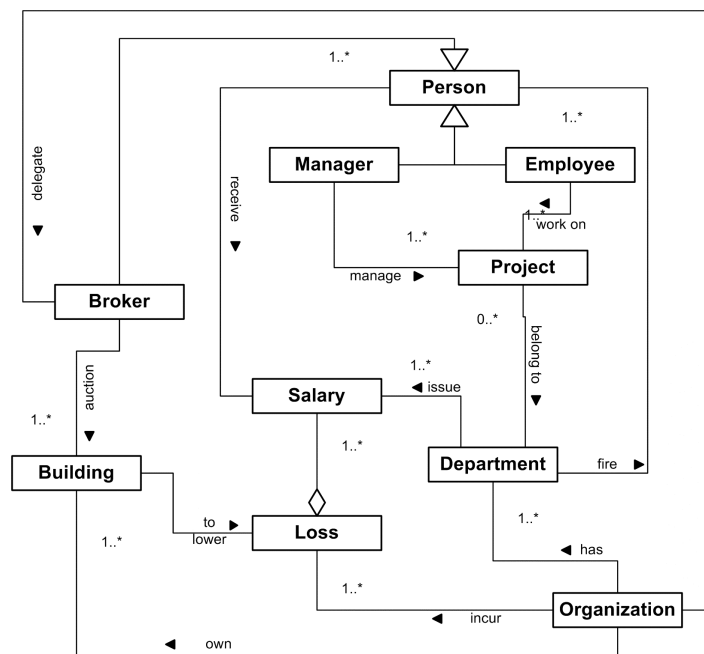


Figure 4 - Traditional Model for the pattern

Comparison:

Table 1 - Comparison between Traditional Model (TM) and Stable Pattern (SP) based on Modeling Adequacies

1	Adequacy:	Descriptive Adequacy	
Descriptive adequacy refers to the ability to visualize objects in the models. Every defined object should be browse-able, allowing the user to view the structure of an object and its state at a particular point in time. This requires an understanding and extracting meta-data about objects that will be used to build a visual model of objects and their configurations. This visual model is domain dependent -- that is, based on domain data and objects' meta-data.			
<i>Comments and Measures</i>		<i>Adequate</i>	
TM	The traditional model class diagram describes the system in a way that shows which class is interacting with which other class and the relationship between them, but fails to address the goals of the system and as to why this is happening, giving a very static view which doesn't show what is being done for what.	50%	
SP	The Stable Pattern clearly shows the various themes of the system and the BOs that realize them along with the IO classes pertaining to the system that is being implemented. It also has all the information that the traditional model had for describing the relations between the classes and the various conditions and constraints. It describes the system more effectively than what Traditional Model did. It is a good 'in a nutshell' view of the system; however a textual description of the entire system is probably a better way to describe the system.	90%	GOOD

2	Adequacy:	Logical Adequacy	
Logical adequacy refers to the representation tools that describe the pattern's participants' behavior, roles and responsibilities.			
<i>Comments and Measures</i>		<i>Adequate</i>	
TM	Describes the classes and relationships, but only of one scenario that the pattern is catering to, so is a very static view of the system and does not show which class is fulfilling which responsibility.	30%	
SP	Describes the classes and relations of the pattern as well as the application being catered to by showing how each IO is connected to each other forming the application, and how each of them is connected to a BO, which realize the EBTs, giving a more logical understanding of how the system is functioning. However it is still not a dynamic view.	90%	GOOD (Much Better)

3		Adequacy:	Understanding Adequacy
Understanding adequacy relates to the easiness with which it is able to understand for anyone, how the system is built and what it is all about; how easy is it to understand.			
<i>Comments and Measures</i>			<i>Adequate</i>
TM	The class diagram is visually self describing in a way that it shows the classes and the associations clearly. There are no cross lines and the flow is easy to understand. The system can easily be understood by this however, only the static and application specific view of it. It does not still show the main aim and the concept of business that the system is made for.	60%	
SP	The class diagram of the stability model is very effective, when it comes to understanding what the system is doing and how it is doing so. It has all the features that the traditional model provided, but also goes ahead to explain the aim(s) or the goals that the system is supposed to accomplish and the way the same is being done in the application that is in question. One can get to know how the concepts get implemented finally through the IOs or how the IOs are able to meet the goals that the system is supposed to achieve.	100%	GOOD

4		Adequacy:	Analytical Thinking Adequacy
Analytical thinking adequacy relates to the analytical thinking approaches and tools that are concentrating on analytical aspects of the system. It also includes the utilization of analysis patterns			
<i>Comments and Measures</i>			<i>Adequate</i>
TM	Traditional model class diagram is not analytically adequate at all. One cannot analyze the system in any way using this. The goal behind doing anything and where a class fits into the big picture of achieving one or more ultimate goals cannot be judged from it.	10%	
SP	Stable Pattern class diagram is very adequate, when it comes to analytical thinking adequacy, as it shows very clearly the exact place of any class in the system, what the system is built for and how in the big picture a class fits in towards realizing the ultimate goal the EBT. It uses clear and appropriate use of Stable Analysis and Stable Design Patterns to do this	95%	GOOD (Huge Improvement)

5		Adequacy:	Systematic Adequacy
Systematic adequacy relates to the systematic approaches that are utilized in modeling, such as bottom-up, top-down, and middle-out approaches. It also includes functional decomposition techniques, and the selection of the correct level of abstraction at each stage of analysis.			

<i>Comments and Measures</i>		<i>Adequate</i>	
TM	The traditional model class diagram has no concept of taking a very systematic approach to modeling the system. It is a view of the system showing how things are and it doesn't go from one stage of analysis or abstraction to other.	10%	
SP	The Stable Pattern class diagram is very systematic. There are the ultimate goals of the system that are realized through the BOs and they are finally implemented through the IOs, going from top to bottom in a systematic approach to modeling the system. It clearly shows what the goals of the system (EBTs) are and how they are realized through respective BOs and then the IOs that hook up to the BOs.	100%	VERY GOOD (Way ahead)

6	Adequacy:	Epistemological Adequacy	
Epistemological adequacy refers to tools for representing objects in the real world.			
<i>Comments and Measures</i>		<i>Adequate</i>	
TM	The traditional model is an application specific view of the system and is very static. There are no levels of abstraction anywhere except for the inheritance or such relationship e.g. roles of any person etc. not much adequate.	10-20%	
SP	The Stable Pattern class diagram shows the various levels of abstraction of the system i.e. the ultimate goals or the EBTs, then coming down to the next level of abstraction i.e. the BOs which show how the EBTs are realized and then finally the IOs, so the application is abstracted out into different levels and it presents a much better view of the system.	90-95%	VERY GOOD (Much better)

7	Adequacy:	Simplicity Adequacy	
Simplicity adequacy relates to how simple your models will be.			
<i>Comments and Measures</i>		<i>Adequate</i>	
TM	The class diagram shows the classes that have been included in the use cases. The flow and associations are clearly mentioned in the form of arrows and lines. This makes it simple to read and follow.	85%	
SP	In terms of simplicity, the Stable Pattern class diagram shows the classes and the relationships between them. The direction arrows show the flow quite well. It is close to traditional model in terms of simplicity. The only thing that makes it better is the holistic view of the system, showing the EBTs, BOs and IOs and also showing them in neatly segregated sections increasing the simplicity.	99%	GOOD

6.2. Measurability

6.2.1. Quantitative

Sl. No.	Criterion	Traditional Model	Stability Model	Indicator (Aspect w.r.t. Stability Model)
1	Number of Aggregation Relationships	1	4	Good
2	Number of Inheritance relations	3	2	Good
3	Number of Applications	1	Many	Good
4	Number of Tangible Classes	10	8	Good

Table 2 - Quantitative Comparison of Traditional and Stability Models

Implications of the above mentioned indicators:

Number of Aggregation Relations & Inheritance Relations:

As we can see, the traditional model is not very flexible and adaptable to more than one application. The number of inheritance relationships is more in the traditional model than the number of inheritance relationships in the stability model, which makes it evident that the traditional model makes its classes, especially the sub classes that inherit from their super classes, very brittle. This makes the system difficult to maintain and has a lot of dependencies in it. If a super class has to be modified for some reason, it affects all of its base classes and higher the number of inheritance relationships, the more impact the system will undergo. The Stability model reduces this and has more number of aggregations in it than the traditional model, thereby giving the system a flexibility of modifying just those parts which are necessary, without propagating the effect further into the system, thus making it more stable over time and easier to maintain. It also results in a significant reduction of maintenance costs, which currently forms more than 80% of the cost of any software.

Number of Applications:

Traditional Model is a static model and it is the view of a particular scenario, and a specific application; it is impossible or very hard and too costly to adapt it to any other application, hence it just serves one application. Stability model on the other hand, has the ability to be modeled to any number of applications, because it has been modeled on the general concept of EBT and Maintenance in this case, and it can be applied to any other scenario as well; hence this is a good feature that stability model offers.

Number of Tangible Classes:

In a traditional model, the number of tangible objects used is 10, when compared to 8 in the stability model. This speaks for itself in the lack of flexibility of the traditional model and the cost of maintenance. Requirements always change over time and trying to adapt to the changing needs, the traditional model does not satisfy that need, since a large part of the system would have to be changed for that, whereas in the stability model the impact would be considerably lesser than that in the traditional model, since we just need to modify the IO classes and we are at ease to go forward.

6.2.2. Qualitative

Adaptability & Reuse

The Software Stability model is highly adaptable to changes in business needs and in the system. The large number of inheritance relations in the traditional model makes it very difficult to maintain and change according to the changing needs. This problem is reduced in the Stability model due to lesser inheritance relations. The changes and modifications to be made to a super class are not propagated further into the system to the subsequent sub classes, hence making it easier to maintain, adapt and reuse. The traditional model also does not have the capability to be modeled to a different scenario, while the stability model owing to the approach of modeling the system on the lines of an EBT i.e. Maintenance in this case, and BOs, gives us the capability to be modeled to any other scenario, where maintenance is required by AnyCorrectiveAction, thereby making it domain independent.

7. Modeling Issues, Criteria, and Constraints

7.1. *Abstraction*

To design this project, we have used the software stability model, which consists of three parts. The Enduring Business Theme and the Business Objects form the essential parts of the model. They remain stable throughout any design and for any application. The IOs change for different applications. The EBT is the Enduring Business Concept, it is the Stable Analysis model and using this ensures that the ultimate goal of the system or business as such is never compromised, and that the system is modeled using that goal in mind. The EBT is then realized by using BOs or Business Objects, which are the workhorses for this EBT and these BOs provide hooks various IOs or Industrial objects to be attached to the pattern model according to the system that we are planning to design.

Here, the EBT is Maintenance, which serves as the goal for any such system and there are various BOs associated to it like AnyCorrectiveAction, AnyPolicy, AnyMechanism, AnyEvidence, AnyLog etc. they realize the enduring concept of Maintenance and according to the application that we want, e.g. reducing losses to an organization in this case, we attach the necessary IOs like Employee, Company, Loss, Policy etc. to these BOs, thereby modeling the entire system that is wanted.

7.2. *Modeling Heuristics*

While modeling the system, we need to follow certain modeling heuristics that enable a better system design. Few of them are as follows:

No Star: There should not be any macho classes in the system, since having a star in the system calls for a focus of control on one class and it gets overloaded making the system prone to failures because of it. In our system, classes such as AnyCorrectiveAction and AnyCriteria have three connections already in the pattern class diagram, so we need to make sure that while connecting the IOs, we do not overload these classes by having more than one new connection to it, as that would result in an unbalanced situation and a macho class or a star.

No Dangling: No dangling classes should be left in the system. Dangling refers to a situation, where there is just one connection to a class. This situation is not good, since it creates a possibility of loss of control, as one the execution reaches that class it cannot be brought back. In this system, we have AnyEvidence dangling in the pattern class diagram, thus we need to make sure that we have at least one IO that is connected to this BO, so that AnyEvidence is not left in a dangling state.

No Sequence: There should not be any sequence because it results in many situations where the system may go into infinite loops without resulting in any tangible output. The figure below shows such a situation.

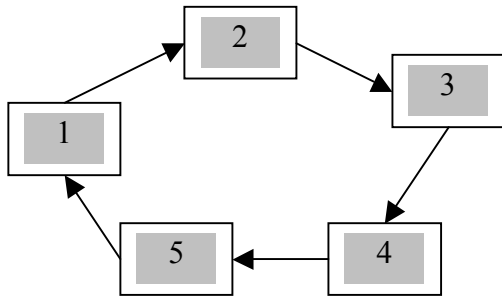


Figure 5 - Sequence (MUST be avoided)

Moreover, if there is a sequence, then it may also mean that the selection of components and classes was not proper. Thus, when there is a sequence of say 8 classes, it may rather be clubbed into one whole class or 2 classes that are communicating with each other, as shown below:

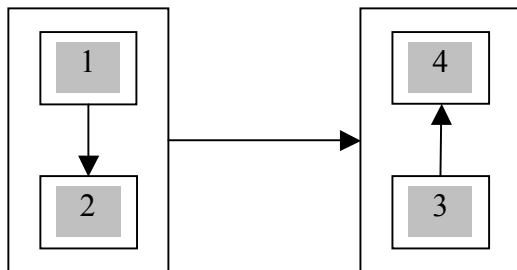


Figure 6 - Alternate view of an existing sequence

General Design: The system should not be designed, while keeping in mind just one domain or application. We are looking at designing a system, which can be reused readily and can be applied to any domain in a domain independent fashion. Thinking about the possibilities in just a particular scenario will limit the capabilities of the pattern and will defeat the purpose of Stability Modeling.

8. Design & Implementation Issues

8.1. Design Issues

Selecting EBT & BOs:

To accurately model the Stable Design Pattern of AnyCorrectiveAction, the first and the most essential part is to choose the correct EBT for it. There are instances and goals which can be confused with the ultimate goal of a corrective action, like instead of choosing Maintenance,

one may get confused with Betterment or Rectification, but AnyCorrectiveAction is not used for those situations, which require betterment as such. Maintenance is the correct EBT for the system. Also, the related BOs are very important as well. The BO AnyCorrectiveAction cannot realize the goal of Maintenance, without the presence of the correct and other relevant business objects in the system. AnyCorrectiveAction must be approved by AnyParty and must have AnyReason for it. The party is supposed to approve the corrective action. Then, AnyCorrectiveAction must produce AnyLog and AnyEvidence to support and show that the corrective action was taken and accomplished, so that the issuing authority gets to know that it was done. Hence, the correct selection of the EBT and other BOs in the system is critical for the correct behavior of the pattern.

8.2. Implementation Issues

Using Command Pattern:

While implementing, instead of using approaches like inheritance etc., which make the system too brittle owing to the property of the methods of super class being transferred to the sub class even if some of them are needed or not, we can use the Command pattern to implement the system and this can be used while linking BOs to IOs.

In the Command pattern, there is a Command Interface and a ConcreteCommand implements this interface. There is a receiver that contains the specific method's code and is called by the ConcreteCommand upon request from the invoker, who wants to invoke that specific method. This allows us the flexibility of not having the clients tie to specific implementation code and they can just use the command to call a function without having to bother about its location of implementation details. On the service provider's end, it provides the system the flexibility to be maintained in a way that just the specific parts of the system are changed without affecting other classes in the system or the clients.

Here is the basic structure of the Command Pattern:

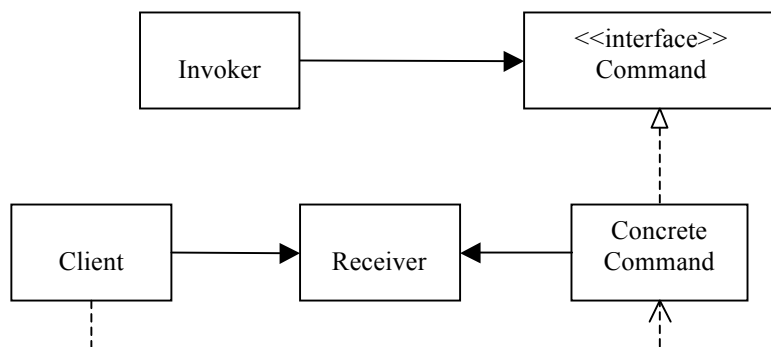


Figure 7 - The Command Pattern [8]

In our pattern the Command pattern can be applied as follows:

```

/*the Command interface*/
public interface Command{
    void execute();
}
  
```



```

/*Receiver class*/
public class AnyCorrectiveAction{

    public Counseling(){ }

    public void produceLog(){
        System.out.println("producing log");
    }

    public void resultEvidence (){
        System.out.println("making evidence");
    }
}

/*the Command for producing log*/
public class ProduceLogCommand implements Command{

    private AnyCorrectiveAction myCorrAction;

    public ProduceLogCommand(AnyCorrectiveAction corrAction){
        this.myCorrAction =corrAction;
    }

    public void execute(){
        myCorrAction.produceLog();
    }
}

/*the Command for resulting evidence*/
public class ResultEvidenceCommand implements Command{

    private AnyCorrectiveAction myCorrAction;

    public ProduceLogCommand(AnyCorrectiveAction corrAction){
        this.myCorrAction =corrAction;
    }

    public void execute(){
        myCorrAction.resultEvidence();
    }
}

/*the Invoker class*/
public class Invoker {

    private Command evidenceCommand;
    private Command logCommand;

    public Invoker(Command evidenceCmd, Command logCmd){
        this.evidenceCommand=evidenceCmd;
        this.logCommand=logCmd;
    }

    public void resultEvidence(){
        flipUpCommand.execute();
    }

    public void produceLog(){
        flipDownCommand.execute();
    }
}

```

```

/*The Client*/
public class Client{

    public static void main(String[] args){
        AnyCorrectiveAction corrA = new AnyCorrectiveAction();
        Command ev=new ResultEvidenceCommand(corrA);
        Command lg=new ProduceLogCommand(corrA);

        Invoker correctiveAction=new Invoker(ev,lg);

        corrA.produceLog();
        correctiveAction.resultEvidence();
    }
}

```

9. Business Issues

Business Rules

Business rules are constraints that govern the way in which a business is carried out. Our system pertains to the ultimate goal of Maintenance, which is achieved through AnyCorrectiveAction and other related BOs. Here are some business rules that apply to this system:

- a. AnyParty can define one or more AnyCriteria. Only specific kind of parties can create AnyCriteria. They are government agencies for administering the country for its well being. Scientists who discover and make theories or laws based on empirical vales and experiments, companies who make them for the employees to abide by for code of business and ethics.
- b. AnyMechanism is controlled by AnyCriteria. No mechanism being employed for corrective can go beyond the boundaries set forth by one or more governing criteria.
- c. There must be AnyEvidence and AnyLog, which results from AnyCorrectiveAction, which is being taken that would serve as a proof to the authority that issued and approved it, of the fact that corrective action was taken.
- d. AnyParty is the governing authority that observes the situation and approves that AnyCorrectiveAction be taken. AnyCorrectiveAction cannot be taken without the permission and approval of the governing authority.

10. Known Usage

- Corrective Action is a term which is more familiar in military. Many of the Military applications use this pattern to carry out many of its tasks; for example, there can be a situation which involves the military to reroute or reframe its strategy of attack. This has to be authorized by the commanding authority and the corrective action of realigning the military to organize a coordinated and effective plan of action, will be taken.
- The pattern has also found its implementation in the business world. “A corrective action is a change implemented to address a weakness identified in a management system. Normally corrective actions are implemented in response to a customer complaint, abnormal levels of internal nonconformity, nonconformities identified

during an internal audit or adverse or unstable trends in product and process monitoring such as would be identified by SPC.”[9]

- Any Company can use this pattern to correct any behavior that leads to some sort of non conformity to its rules and policies.
- Another known usage of the pattern is among educational institutions like universities to take corrective actions in order to counter the problems like plagiarism or discrimination etc.
- The pattern can really be applied to any situation or domain, where there is a need for Maintenance, due to some reasons of non conformance to the desired behavior. Though traditionally used in military applications, it can be exported over to other domains as well.

11. Tips and Heuristics

- Betterment, as good as it may sound to be an EBT for AnyCorrectiveAction pattern, actually it isn't. Maintenance is a better choice.
- To fit the EBT of Maintenance and to realize the pattern AnyCorrectiveAction, the other BOs in the system must be appropriate and in context with AnyCorrectiveAction or else the true purpose of this may not be solved.
- There were certain BOs chosen earlier, but discarded later, because they did not fit in the pattern or were redundant; also, it was hard to find IOs for those BOs.
- While developing the IO, when the connections are not possible, it means that something is missing in the BO or the chosen EBT is not correct. This situation was faced, when a particular BO was never used in any application or an application could not be linked.

12. Conclusion

Stability modeling approach towards modeling the system is a big leap ahead, from the Traditional way of modeling the system, because it provides us a pattern that can fit more than one, in fact many applications, in a domain independent fashion. While in traditional modeling, we developed a system with its specific purpose in mind and modeled just one application of a pattern, it was not possible to model to any other scenario. But, the stability model keeps in mind the ultimate goal, and it models a system accordingly, thus making it a flexible and a more maintainable system. Here, we saw that corrective action is a technique applied mostly in military. However, when we model it by using the stability modeling technique, we can take it and apply it to domains like business as shown in the applicability part. It was a good learning experience and an eye opener to realize the flaws that existed in the current approaches and to experience a better way of software modeling.

13. References

-
- [1] “Corrective Action”, n.d. [Online]. Available: Onelook.com <http://onelook.com/?w=corrective+action&ls=a> [Accessed May 26, 2010].
 - [2] M. E. Fayad, and A. Altman, “Thinking objectively: an introduction to software stability”, Communications of the ACM, vol. 44, no. 9, pp. 95, Sep. 2001.

-
- [3] H. Hamza, A. Mahdy, M. E. Fayad and M. Cline, "Extracting Domain-Specific and Domain-Neutral Patterns Using Software Stability Concepts", Lecture Notes in Computer Science, vol. 2817. Berlin / Heidelberg: Springer, 2003, pp. 191-201.
- [4] "Correction", n.d. [Online]. Available: MSN Encarta, <http://encarta.msn.com/encnet/features/dictionary/DictionaryResults.aspx?refid=1861600494> [Accessed May 26, 2010].
- [5] "Correction", n.d. [Online]. Available: YourDictionary.com: <http://www.yourdictionary.com/correction> [Accessed May 26, 2010].
- [6] M.E. Fayad, H.A. Sánchez and H.S. Hamza. A Pattern Language for CRC Cards. Proceedings of Pattern Language of Programs' 2004 (PLOP'04), Monticello- Illinois, USA, Sept. 2004.
- [7] M.E. Fayad, Haitham Hamza, and Huáscar A. Sánchez "A Pattern for an Effective Class Responsibility Collaborator (CRC) Cards". Proceedings of the 2003 IEEE International Conference on Information Reuse and Integration (IRI'03), Las Vegas, Nevada, pp. 584-587, October 2003.
- [8] "Command Pattern", n.d. [Online]. Available: Wikipedia, http://en.wikipedia.org/wiki/Command_pattern [Accessed May 26, 2010].
- [9] "Corrective Action", n.d. [Online]. Available: Wikipedia, http://en.wikipedia.org/wiki/Corrective_action [Accessed May 26, 2010].