2004

Pattern language for risk assessment, analysis, and management

Elham Yavari
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Yavari, Elham, "Pattern language for risk assessment, analysis, and management" (2004). *Master's Theses*. 2701.

DOI: <https://doi.org/10.31979/etd.bwwd-7gpa>
https://scholarworks.sjsu.edu/etd_theses/2701

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

**PATTERN LANGUAGE FOR RISK ASSESSMENT, ANALYSIS, AND
MANAGEMENT**

**A Thesis Presented to
The Faculty of the Department of
General Engineering
San Jose State University**

**In Partial Fulfillment
Of the Requirements for the Degree
Master of Science**

**By
Elham Yavari**

August 2004

UMI Number: 1427070

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI Microform 1427070

Copyright 2005 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

© 2004

Elham Yavari

ALL RIGHTS RESERVED

APPROVED FOR THE
DEPARTMENT OF ENGINEERING

Professor Mohamed Fayad

Professor Dan Harkey

Dr. Hamid Rategh
(Director of Engineering, Anadigics Inc)

APPROVED FOR THE UNIVERSITY

ABSTRACT

A PATTERN LANGUAGE FOR RISK ASSESSMENT, ANALYSIS, AND MANAGEMENT

By Elham Yavari

Considering the wide range of applicability of Risk Assessment, Analysis, and Management (RAAM) systems, this thesis intends to provide a system of stable patterns to capture the core knowledge in RAAM and organize this area's main concepts in an understandable, reusable, scalable pattern language. Therefore, the guidelines of Software Stability Modeling (SSM) are followed. As a new method for identification and design of stable patterns, SSM describes for any pattern language different pattern sources such as missions and goals, capabilities, architectures, deployment, and development. During this work, twelve stable analysis patterns as well as thirty stable design patterns are determined. Moreover a knowledge map is developed that associates the patterns and introduces different frameworks in the area. Furthermore, three patterns are selected and described in more detail in a few examples. Moreover some application instances are implemented to illustrate the applicability of those patterns in different application areas.

ACKNOWLEDGEMENTS

I appreciate the following people for their help and encouragement during this work:

Dr. Mohammed E. Fayad, for his guidance and support and for his great way of teaching;

Professor Dan Harkey and Dr. Hamid Rategh, for their valuable comments and advice;

All my friends in the pattern-group for their precious assistance and company;

Haritha Kilure, for her kind help in pattern implementation;

And my husband Ali, for being so understanding and helpful during this work as always

Table of Contents:

Chapter 1- Introduction	1
1. Research Proposal	1
1.1.What Is Risk?	2
1.2.What Is RAAM?	3
1.3.What Are RAAM Benefits?	5
1.4.What Has Been Done In RAAM Area Before?	5
2. Research Methodology	6
2.1.Software Stability Method (SSM)	7
2.2.Stability Based Pattern Language	10
3. Thesis Structure	14
4. How To Conduct The Research	14
5. My Contributions	16
6. Conclusions	17
 Chapter 2 - Goal-Oriented System Analysis	 18
1. First Level Goals	20
1.1.Predictability	21
1.2.Planning	21
1.3.Reliability	22
2. Second Level Goals	23
2.1.Risk Identification	23
2.2.Measurement	24
2.3.Recording	26
2.4.Allocation	27
2.5.Alternative Management	28
2.6.Risk Reduction	28
3. Conclusions	29
 Chapter 3 - How To Achieve The Goals	 30
1. Explicit Capabilities	32
1.1.AnyProject	32
1.2.AnyTask	33
1.3.AnyMilestone	34
1.4.AnyResource	34
1.5.AnyDependency	35
1.6.AnyRisk	36
1.7.AnyRiskIdentificationMethod	37
1.8.AnyMetric	37
1.9.AnyProbability	38
1.10. AnySeverity	38

1.11.	AnySensitivity	38
1.12.	AnyLog	39
1.13.	AnyEntry	40
1.14.	AnySchedule	40
1.15.	AnyAlternative	41
1.16.	AnyCost	41
1.17.	AnyCriterion	42
1.18.	AnyAcceptation	42
1.19.	AnyAvoidance	42
1.20.	AnyTransferring	43
1.21.	AnyMitigation	43
2.	Common Capabilities	43
3.1.	AnyParty	44
3.2.	AnyMedia	44
3.	Implicit Capabilities	45
3.1.	Fault Tree	46
3.2.	Cut Sets	46
3.3.	Mission Timeline	46
3.4.	Reliability Block Diagram	47
3.5.	Event Sequence Diagram	47
3.6.	Component diagram	47
3.7.	Binary Decision Tree	48
4.	Conclusions	48
Chapter 4 – Architectures In Demand		49
Chapter 5 - From Design To Implementation		56
Chapter 6 - Setting The Main Direction For Quality Assurance Activities		60
1.	Reusability	61
2.	Scalability	62
3.	Understandability	64
4.	Tracability	64
5.	Concolutions	66
Chapter 7 - Wrapping It Up		67
1.	Stability-Based Pattern Language	68
2.	Contributions	70
3.	Future Work	71
References		72

Appendices:

A. Pattern Catalog	75
1. AlternativeMangement Analysis Pattern	75
2. AnyProject Design Pattern	86
3. AnyRisk Design Pattern	99
B. Implimentation	111

List Of Figures:

Figure 1 - Software stability modeling layers	8
Figure 2 - Main activities in providing a pattern language	13
Figure 3 - RAAM system's classification	20
Figure 4 - AND/OR prerequisite relationships	24
Figure 5 - Stable design pattern for AnyProject	33
Figure 6 - Stable design pattern for AnyResource	35
Figure 7 - Four forms of dependency between timed elements	35
Figure 8 - AnyRisk stable design pattern	37
Figure 9 - AnyLog stable design pattern	40
Figure 10- Tasks T1,T2,T3,T4 and tasks T1,T5,T6,T7: two alternatives for the same target	41
Figure 11- AnyParty stable design pattern	44
Figure 12- AnyMedia stable design pattern	45
Figure 13- Knowledge map, architectures and stable patterns	51
Figure 14- The development sequence for stable patterns, knowledge map, and architectures	52
Figure 15- Map for RAAM systems	54
Figure 16- Communication between design and implementation teams	56
Figure 17- Project lifecycle with one testing step at the end	60
Figure 18- Project lifecycle with several testing steps after each phase	60
Figure 19- Deployment map for RAAM systems	65
Figure 20- Stable analysis pattern for AlternativeManagement	78
Figure 21- Detailed pattern structure for AlternativeManagement stable analysis pattern	79
Figure 22- Stable design diagram for path-map problem	83
Figure 23- Stable design diagram for obesity solutions	84
Figure 24- Stable design pattern for AnyProject	89
Figure 25- Detailed stable design pattern for AnyProject	90
Figure 26- Stable design diagram for the Manhattan project	94
Figure 27- Stable design diagram for publishing a paper	95
Figure 28- Sequence diagram for initiation usecase	97
Figure 29- Risk, probability of occurrence, and severity	99
Figure 30- Stable design pattern for AnyRisk	103
Figure 31- Detailed stable design pattern for AnyProject	103
Figure 32- Stable design diagram for a farming project	107
Figure 33- Stable design diagram for a school trip	108
Figure 34- State transaction diagram for risk reduction activities	109

List Of Tables:

Table 1- Different viewpoints in system development and their targets	10
Table 2- System development phases' interpretations	11
Table 3- BOs in RAAM system	32
Table 4- Usecase description template for project initiation	96

Chapter 1

Introduction

This thesis specifies a pattern language for Risk Assessment, Analysis, and Management (RAAM) following the guidelines and rules of Software Stability Modeling (SSM). Although RAAM is a familiar concept, applicable to almost any kind of project, the thesis will present all common aspects of RAAM regardless of any specific application and/or area. Such a general pattern would provide a strong backbone design for reuse in any RAAM application. The study depends on Software Stability Models and Pattern Language techniques that will be described later.

1. Research Proposal

The current thesis explains the main concepts of risk assessment, analysis, and management using a pattern language. This Pattern Language (PL) will be developed as a system of patterns including the core knowledge of RAAM. The PL will be prepared following the cognitive process of understanding and generalizing main concepts of risk management independently from any particular application or area and will serve as a backbone design for any related application. The pattern language will be produced based on the nature of SSM, which is a new methodology to analyze and design more stable systems. Using SSM guidelines, common aspects that all applications with the same purposes and performances share will be identified and described. These aspects of the system will remain stable over time and/or in cognitively similar applications.

1.1. What Is Risk?

Any project faces some uncertain events in its different stages that can negatively affect the project plan. It can lose one of the predefined resources (money, personnel, equipment...) for some reason or may face new requests for deliverables or timetables. Some unpredicted costs such as the cost of new requirements might be also imposed on the project. This thesis addresses such unwanted events that may occur in project and cause damage to its properties or targets by the term: "RISK." [10, 11, 12]

Risk occurs for two major reasons: [13, 14].

- Sometimes the event itself is variable in nature so the risks represent randomness in any sample of the statistic population. Risk managers traditionally refer to this type of event as "Risk." Car accidents may be counted as an instance of such risks.
- Other risky events happen because researchers lack knowledge about some fundamental phenomena. For example, scientists do not know much about the nature of earthquakes. Earthquakes cause a lot of damage that could be avoided if scientists knew more. Risk managers once classified the events of this group as "Uncertainties."

The distinction between these two types of uncertainties is not that significant. Recent works [15] usually refer to both types simply as "risk" mostly because the actions that should be taken to deal with the two of them are similar. Risk is usually defined by the combination of probabilities that a project or other enterprises will experience an undesired event with unacceptable consequences, impact, or severity. For example, if there is a possibility that one of the project team members leaves the team before his deadline and if this would cause technical difficulties or time delay in the project, there is a risk whose value relates to the probability of losing the team member and its overall

severity. Risk managers usually describe both probability and impact of the risk numerically so they can compare and rank possible risks based on their values.

Risks are connected to each other in different ways. For instance, when a particular activity in a project is a prerequisite for another activity any risk that occurs in the first activity can also negatively affect the risks in the second one. The effect would be either to increase the possibility of other risks or to enhance their severity. The risk of having drought or fire in a farm is a good example of this connection between the risks. Although fire can happen independently and therefore should be studied as a separate risk, it is not totally independent from drought. This is because having a drought considerably increases the possibility of fire accidents.

1.2. What Is RAAM?

Even everyday projects require instances of risk management. Actuaries identify the risk of fire accident in our homes. They try to prevent accidents by recommending smoke detectors or reduce risk by suggesting fire fighting systems transferring it to other organizations by insuring properties. Activities like this, which intend to handle the risky situation with minimum exposed cost, are classified in Risk Assessment, Analysis, and Management (RAAM) class of activities. RAAM covers the total process of controlling uncertain events. It is a proactive, continuous and iterative process designed to achieve the planned objectives in terms of money, time, and quality. The process involves identifying, analyzing, planning, tracking, controlling, documenting, and communicating risks effectively [1, 7]:

- **Assessing risks:** It is a continuous effort to identify and document risks as they are found. It covers all different types of risks such as technical, legal, financial, human, and environmental risks. Using Contributing-Factor Diagrams and studying each project component deeply are the main steps in risk assessment.

- **Analyzing risks:** It is necessary to estimate the probability, impact, and timeframe of the risks. Managers also need to classify sets of related risks and them based on their type, impact, probability, and timeframe. There are different methods for risk analysis such as Frequentism or Bayesian methods, which measure the risk factors based on either the previous experiences in similar projects or the subjective evaluation of the experts in the field.

- **Planning:** Then the decisions are made about what to do with the risks. These decisions include ignoring, mitigating, transferring, and avoiding plans. The decisions are made based on the results of risk measurements from the analysis phase.

- **Tracking and controlling risks:** In order to take the best actions as soon as a risk happens, risk engineers need to collect and report status information about risks and their treating plans. In order to organize the efforts in this phase, RAAM system will assign each risk, or each group of related risks, to a risk officer and he or she will be responsible for the risk custody activities.

- **Communicating and documenting risks:** RAAM process continues at each step of the project to provide required support for decision-making process. Occurrence of a risk as well as the mitigation actions taken to control it and the result damage must be reported to all involved persons and should be documented in RAAM database. It is more important for the systems that use historical data for risk evaluation (Frequentism).

1.3. What Are RAAM Benefits?

RAAM is a complicated task and needs special time, personnel, and monetary resources. However, it has different benefits for the project:

- Reducing the probability and the impact of risks and therefore increasing the system's reliability
- Gaining and keeping DAA (Designated Approving Authority) approval
- Supporting a better decision making process by predicting detailed aspects of risks and their impacts
- Planning for a better resource allocation process and controlling the project continuously

1.4. What Has Been Done In RAAM Area Before?

Two different types of work clarify and utilize the principles of RAAM in different areas of software development: "special interest tools" and "setting standards." A growing interest in risk management encourages the development of exciting new tools and services. As a result, a broad array of established risk management tools are ready to be used. "Risk Management Tool for EU Agriculture" [16] and NASA Continuous Risk Management [17] are instances of such applications. However, these software systems concentrate on a specific organization or at most a restricted domain of projects. The fact that they are restricted to special problems encourages engineering managers to seek more general tools that capture the common core knowledge of risk management independently. Item Software QRAS [18] is an example of the efforts towards a general RAAM tool.

Another major work is to establish some standards like the “Australian standard.” The joint Australian/New Zealand Standard AS/NZS 4360, first published in 1999, addresses the process that should become an integral part of any strategic plan of any organization [19]. This process includes several steps:

- Development of an organizational risk management philosophy at senior management levels by training, educating and briefing of executive management
- Assignment of the responsibility for advising the chief executive officer/governing body on risk management issues to a particular risk officer
- Development of reference terms for the Risk Management Committee and establishment of the role and mandate of the RAAM Committee in the organization as well as the clarification of its relationship with the chief executive officer/governing body
- Development and documentation of a corporate policy for RAAM, to be implemented throughout the organization
- Establishment of a general culture of RAAM that involves all staff and processes
- Regular monitoring and reporting on the level of risk exposure and tracking changes in the activities and operations of an organization that may introduce new risks or eliminate others

The standard also has classified important risks of various areas, which is very attractive for business owners as well as for engineering managers. However, they also have the problem of restricted domains.

2. Research Methodology

This thesis will use Software Stability Method [2, 3, 4, 5] to define a pattern language for RAAM in general. Hamza and Fayad [8, 9] also illustrated the use of

software stability to create pattern languages for other domains as instances. The following section will demonstrate the use of SSM and its algorithms that will be applied to format the general information to make stability based pattern languages:

2.1. Software Stability Method (SSM)

Ordinary design methods usually build the system based on its current requirements. Therefore, the applications are very sensitive to any future changes in user requirements so the maintenance phase is very costly for today's software systems. On the other hand, because the requirements include a lot of information about the ongoing project and its application domains, requirement-based software systems are highly dependent on a specific application and domain area. Any reuse in other domains would be difficult and expensive. The costly requirement-engineering phase is another main problem in traditional software design methods. It also usually does not cover all users' needs because it mainly depends on the customer's ability to understand and notify system requirements.

SSM is a new modeling technique that makes software systems based on their core concepts and our general knowledge of them. This way we ensure that our systems are general enough to accept further requirements and to be reused in any application with similar goals and concepts. The classes in this methodology are classified into three different layers. Requirement engineering and preliminary design phases are done at the same time that a designer seeks these three class types:

1) Enduring Business Themes (EBTs) represent the main purposes of the system. These classes, in fact, capture the core concept for which we made the system. This layer

corresponds to “requirement analysis objects” which are called “Stable Analysis Patterns.”

II) Business Objects (BOs) form the concrete classes of the system that are necessary for system to fulfill its goals. BOs are generally called the “work horses” of the system and show the essential and general parts of any system with the same EBT. This layer corresponds to “design objects” which are called “Stable Design Patterns.”

III) Industrial Objects (IOs) represent the particular applications that can be hooked to the pattern. Information in these tangible classes is specific for an application or a domain. IOs usually can be replaced without affecting the system. This layer corresponds to “application or implementation objects.” The former three levels of the classes are in the same order in terms of generality and application independence. They capture the core concept of the system in order as shown in Figure 1.

Figure 1- Software Stability Modeling Layers

Because pattern development uses a layered approach, it is important to understand these layers and how these layers are obtained. In order to obtain EBTs of the system, those aspects of the system that remain stable over a period of time are identified. These aspects form the core knowledge of the system based on what the system does and why

the system is modeled. BOs, on the other hand, are engineers' answers to the question of what the system needs to reach its purposes. They are stable classes that may be adapted internally but remain externally the same. They are generally required in any application with the same goals, but have some dependencies on the current technology or frameworks. With the changes in these aspects of the system, we may need minor adaptation inside BOs that are local and invisible for the class's clients. So external stability is provided even in adaptation cases.

As a result of obtaining these classes, the majority of the engineering actions taken on a system modeling concentrate on the stable areas. This yields a stable core design and thus, a stable software product. Changes introduced to the software product will then be in periphery because the core would remain stable. Thus, the endless circle of reengineering the entire system is avoided for minor changes and this reduces cost and time of the changes in the system when the requirements change.

Stable analysis and design patterns are conceptual models that are obtained as a result of capturing the core aspects of the problem under consideration. Patterns may vary in the level of abstraction and it is important to extract features that are useful in handling problems of multiple domains, so anyone can successfully reuse the same model in similar problems. [2, 3, 4, 5]

This thesis is fully developed based on the philosophy of SSM that is followed carefully in all chapters. Main EBTs and BOs of the system are identified in Chapters 2 and 3. There is an overall map in Chapter 4 that graphically shows them as well as their relationships. Although a particular application is not discussed, some examples of

potential reuses in pattern templates are included in Appendix A. IOs and their role of customizing a general pattern for a specific application are contained in those examples.

2.2. Stability Based Pattern Language

Software Management Techniques cover a wide range of activities to use the available sources optimally and gain the best possible results during software system development. All involved parties including customers, end users, engineers, and marketing officers have their own demands and techniques to improve both the process and the product with respect to their own quality factors.

	Viewpoint	Target
1	Software lifecycle	Complete SW Development Methodology
2	Business vocabulary	Dynamic Analysis Using The Business Language
3	Software stability	Providing Stability Models for classes and patterns
4	Patterns	Building Systems of Patterns
5	Research	Providing A Pattern Language

Table 1- Different viewpoints in system development and their targets

Any new methodology in this field should address these needs and justify its suggested activities in appropriate forms so each participant can understand the mission and meaning of each proposed activity. As a new software development methodology, Stability Based Pattern Language is not an exception either. It is necessary to define and justify all phases in such terminologies that participants can understand and use. The above table shows some important viewpoints related to stability based pattern language along with their goals and targets.

Stability Based Pattern Language includes some step-by-step guidelines that describe a process for software development. Table 2 shows these phases along with their interpretation in each viewpoint.

Business vocabulary	Software lifecycle	Software stability	Patterns	Research
Goals and Missions	Analysis	EBTs	Stable Analysis Patterns	Classification/ Categorization
Capabilities	Design	BOs	Stable Design Patterns	Properties/ Criteria
Many Architectures	Architecture Design	EBTs-BOs	Architecture Patterns	Map/Many Directions
Scenario development	Implementation	IOs	Process Patterns	Development
Recommendations (Quality factors)	Testing	EBTs-BOs	Analysis, Design, and Architecture Pattern	Deployment

Table 2- System development phases' interpretations

The following sections are devoted to more descriptions and some examples for each step presented in the above table. For more descriptions see reference [6].

- **Classification:** In the first phase, we intend to divide the problem into separate categories to help parties understand and manage them effectively. The classification would be based on system's general goal and missions so each category presents a distinguished goal that is equivalent to one core concept in the field. Classification provides the main EBTs of the system that categorize the problem into conceptually separate classes. In order to find proper EBTs, the problem should be considered conceptually and analyzed to capture the core knowledge independently from particular applications or domains.

- **Properties:** Any system needs a set of capabilities to fulfill its mission. These capabilities are general utilities for EBTs to do the job and make the goals practically achievable. In fact, a capability is a property or tool that is used by an EBT as a workhorse, which is very similar to our definition of BOs in the previous section. The capabilities are described as intermediate concepts that relate abstract EBTs to tangible

IOs in many application areas. Because of this important role for BOs in development of a stable design, they are called stable design patterns.

- **Map and Architectures:** The patterns and classes of the system have been studied more individually in last two phases. The relationships among the classes and an architectural view must be considered in such an order that engineers can easily understand how the classes and patterns work together. Therefore, stability based PL recommends that engineers generate a map containing many directions, each representing a unique architecture. These architectures would be the main baselines for applications' design, which would reuse the pattern language by utilization of its developed patterns.

- **Development:** Now it is time to think about implementation considerations and provide some process patterns to support application development. Process patterns are algorithms that are common in all applications in the area and teach the engineers how to do the processes. They will serve as guidelines and will not be present in the final design as separate classes. Process patterns are powerful tools for documenting the main procedures in the system precisely and will help a lot during the implementation phase. They are strongly related to IOs, which are domain-related tangible classes that customize a model for a specific application.

- **Deployment:** Stability based pattern language strongly recommends the engineers to do validation and verification tests at the end of each phase to make sure that the deliverable products of each one is provided exactly as they were supposed. However, it has also a separate testing phase to emphasize more on the quality factors and the recommended ways to obtain them. In order to identify deployment patterns, any special

consideration that may affect our preliminary design should be considered. Scalability, reusability, adoptability, stability, and understandability are some examples of these issues. Enough description in the form of more EBTs and BOs is provided to show how these quality factors are achieved and kept in later applications.

- **Pattern Templates:** After deciding about main patterns in the stability based pattern language, we need to describe each pattern in detail. After providing a precise definition of a pattern, the description offers the main obstacles for the pattern as well as the provided solution for them. Using CRC cards and dynamic behavior diagrams, the solution would be discussed in detail. The description also shows some applicability examples to increase the pattern's understandability even for beginners. Finally, the positive and negative sequences would be discussed so engineers can evaluate the pattern before any reuse. Figure 2 shows the methodology's scheme briefly:

Figure 2- Main activities in providing a pattern language

3. Thesis Structure

The dissertation includes seven chapters and three appendices. After the introduction, it describes how the stability based PL development is applied to RAAM systems step by step. Chapters 2 to 6 are dedicated to these steps: classification, capabilities, architectural patterns and map, development, and deployment. Each chapter will start with a fairly brief explanation about the related step of PL and follow the application of techniques on RAAM problem. There are summary parts in each chapter to offer their content in one glance. Chapter 7 summarizes all former chapters and talks about the future works that would complete this effort.

Chapters 2 and 3 include some examples of analysis and design patterns, some are explained completely in the standard templates with several application examples to represent how a stable pattern in a real software system can be reused in Appendix A: “Patterns Catalog.” Appendix B: “Pattern Implementations” includes some simple code for example applications based on the pattern templates offered in the first appendix. The last appendix: “Formalization” includes examples of formalized pattern structures that would clarify the patterns. Java as an executable language and OCL as a formalization language have been used for implementation.

4. How To Conduct The Research

Stability based pattern language and SSM are fairly new methodologies, so the next step of this research was to gather enough resources to learn the basic concepts and algorithms. There were heuristics and how-to patterns that I learned by studying developed examples in reference [6]. I described most important points I got during my studies, at the beginning of the chapters, so anyone can use them in any given project.

In order to understand RAAM systems, their commonalities, and the variety of the existing applications, I considered the second phase of this study on RAAM. A lot of good, bad and ugly aspects can be found by the “compare and contrast” method during my revision of the works already done in the field.

Finally I started to develop a pattern language for RAAM by myself. Usually a brainstorming session with the help of my friends in software and reliability engineering fields helped me capture the initial ideas. Then I would review and examine the proposed solutions several times until I was satisfied with the model derived. Afterwards, I tried to apply my solution to some existing application instances to ensure that all necessary aspects are covered. Sometimes I found some modifications helpful to address the application requirements more completely and precisely.

Professor M.E. Fayad is my thesis advisor. He discovered SSM and stability based pattern language, so he is very knowledgeable about the methodology. He has also applied them to different problems, which have given him a good insight about application issues and detailed problems. I was blessed to have his supervision and advice during all stages of this thesis that enabled me to learn numerous useful lessons about this work. We had regular individual and group meetings and email iterations as our communication tools. I need to thank him again for his sincere devotion and interest on this thesis.

The designed patterns in this thesis are implemented using Unified Modeling Language (UML) for modeling, Java for implementation, and Object Constraints Language (OCL) for formalization.

5. My Contributions

This dissertation contains my contributions to the RAAM field that is done under the guidance and supervision of my project advisors. Using the SSM concepts, I have developed a PL for RAAM that includes various analysis, design, architecture, and process patterns for the core concepts of RAAM area. The patterns are gathered and organized in a knowledge map when anybody can study the relationships among them and select proper analysis and design patterns to serve as reusable frameworks. I have also selected some of these patterns and described them fully in standard pattern templates. These patterns are discoveries in themselves and each can be reused in different problem areas. This is because of their generality and independence from any specific area pertaining to RAAM. Being utilized in different applications and demos, each of the patterns has proven to be an actual solution to a recurring problem. Here is a summary of my contributions in this dissertation:

1. The pattern language for RAAM including the main patterns in RAAM area and a knowledge map
2. The following patterns driven from the pattern language and documented in pattern templates:
 - AlternativeManagement analysis pattern
 - AnyProject design pattern
 - AnyRisk design pattern
3. Some application instances for the above patterns implemented in Java and OCL

There still is more work to do to complete this pattern language and to use it in various applications. More patterns are to be developed and documented, hooks are to be

defined and logged, and processes are to be more specified. I listed these major future works in Chapter 7 after I summarized the results of the thesis. However, I am proud of what is done and believe that it is the first step of a long journey.

6. Conclusions

Risk assessment, analysis, and management is a well-known problem in most projects. It deals with identification, measurement, prioritization, documentation, and reduction of project risks and making related decisions like resource allocation and alternative management. These activities are demanded in most projects because they enhance project predictability and reliability, as well as they help managers to design more qualified plans.

This thesis will provide solutions to the problems with contemporary patterns for RAAM and their implementations by developing a stability based pattern language. The developed PL can be used in any domain and encompasses all the essential features to get the system working. It ensures that the pattern is flexible enough to be applied to many domains and remains stable even after addressing new requirements and design changes.

The thesis contribution is briefly a pattern language along with three stable patterns and a few implementation instances to describe the solution, using UML, OCL, and Java as my tools. The major techniques used in this thesis are SSM and stability based pattern language.

Chapter 2

Goal-Oriented System Analysis

Any system is built to fulfill one or more goals, which are the main reasons why we develop the system. They answer the question “why this system is requested?” The first step of PL development is to recognize these missions as the bases of our system. Seeking the main missions in early stages is a feasible task because users clearly know why they initially need the system even though they are rarely sure about all requirements at the beginning. Using goals and purposes as the foundation of the system is very helpful because they are stable over time, reusable in various application areas, and classify the system cognitively.

Because the general purposes of the system are usually stable during the system’s lifetime, we call them enduring business themes or shortly EBTs. One of the most important superiorities of SSM and PL is the fact that they use these stable concepts as the system’s foundation. Using the EBTs enables them to keep the main part of software unchanged during requirement changes and localizes the change in application-dependent Industrial Objects (IOs). This will save money and time when a new requirement is demanded or an existing one is modified. Localization of changes also increases software reliability, which is another benefit of SSM and PL methodology.

Different applications in various areas may have totally different requirements but they have usually common purposes. In traditional methods, where models are based on user’s requirement, software reuse is restricted to the applications from the same area

with similar requirements. However, in the systems based on goals and missions, software reusability covers a wider range of applications with the same core concepts.

Because any possible requirement is requested to achieve a goal, EBTs are proper candidates for categorizing the system into separate categories with strong conceptual relationships among each category. This classification makes the system easy to manage, design, understand, and modify. Using this method, system developers would have a fast access to each subsystem based on its purpose and mission.

In order to recognize the EBTs of the system, the system should be cognitively studied and separated into its main concepts and goals. Some characteristics of an EBT are listed here that can be helpful in the classification process:

- EBTs are stable and enduring. Technical terms or the concepts with usages related to a specific methodology or technique are not good candidates for EBTs.
- EBTs are general enough to be applied in any area with similar core concept. Therefore, limited application-oriented concepts cannot be defined as EBTs.
- EBTs are main concepts in the problem. They are neither tangible nor replaceable. One can find proper EBTs intuitively by understanding the main problem conceptually.
- EBTs show the result of system analysis and generally include the core knowledge of the system. That is why we call them also analysis patterns.
- EBTs, like other parts of PL, will be our system's classes. So they should have unique responsibilities and related properties of their own. If you cannot think of any data or functionality for a candidate it probably would not be an EBT.
- EBTs can be figured out in different levels of generalization. It is useful to have these levels to show the exact boundaries of the system without losing the generality.

- EBTs are aspects of the project. They have the same definition and main characteristics as aspects in the aspect oriented class of project analysis and design.
- EBTs are specifications classes. They specify the regulation of actions in each partition and define the rules and criteria within the category.
- There are other EBTs in the system representing the quality factors, or as most software engineering texts call them non-functional requirements. They define some criteria to verify and validate the result of system analysis and design. Understandability, reliability, and stability are some examples of such EBTs. However, we do not use these EBTs as the main categories of the system. Instead, we have a separate Validation and Verification phase in our pattern language to study the quality factors in more details.

The following sections focus on the RAAM system's concepts and find its main goals and purposes to define them as categories for PL development. See Figure 3 for a brief review:

Figure 3- RAAM system's classification

1. First Level Goals

RAAM systems vary in types and applications. However, the reason for such a system is similar in all fields. Predictability, planning, and reliability are recognized as the high level goals of the system. See Figure1.

1.1. Predictability

Predictability is the ability to predict the project's future to be prepared for proactive actions. It foresees the behavior of a dynamic system based on the knowledge that presently is available about this system.

Having a predictable system is a great blessing for the project managers because they can study about the possible problems ahead of time and take proper actions to fight them best. It also increases their credibility in front of project investigators. That means the investigators believe in the managers' abilities the most, when they see them knowing what is going to happen and what should be considered now to make future problems solved. Another psychological benefit of predictability is to enhance the project participants' self-esteem and motivate them to do their best by making them feel successful and superior.

Predictability is one of the missions of any RAAM system. Having a complete and efficient system for risk assessment, analysis, and management provides highly demanded information about the project for managers that helps them to understand it the best. It is because RAAM systems analyze the whole project from different viewpoints and let the managers know about major risks that may happen during the project lifecycle.

1.2. Planning

Planning is the process of setting goals, developing strategies, and determining tasks and schedules to accomplish particular goals. It includes generation and comparison of different courses of action and then selection of the most suitable one based on the available information. Planning describes various aspects of the system from a high-level

specification of what shall be done to a detailed design of how to do them. It also involves the process of thinking about future decisions and actions.

Planning is one of the most important phases in a project because its quality will affect all other phases in the project. It is done at the beginning of the project life cycle and modified after each phase to customize the plan based on the actual results of the relative phase. A good planning requires a great deal of knowledge and experience as well as a good vision and proper data from the project, so planning has a close relationship with the concept of predictability.

Dealing with the crucial risks is an important part of planning. It starts when we gather all the information in a central database. Then we decide about the resources and how to allocate them to RAAM activities. We also need to decide which one of existing alternatives we want to be a part of the project, based on their costs and involved risks. Finally, we would plan for top priority risks to be dealt with, by using one or more techniques of risk reduction. There are other planning activities outside RAAM system that we did not mention them here.

1.3. Reliability

Reliability is a term constantly used in different areas of human knowledge with almost the same meaning. It is “the probability that a system or product will accomplish its designated missions in a satisfactory manner for a given period of time when used under specified operating conditions.”[24] Reliability has a close relationship to the concepts of validity, trustworthiness, and correctness. However, it is wider than any of them. It is also highly related to the failure rate of the system. In fact, they use mean time between failures (MTBF) as a metric for measuring the reliability.

A system's reliability is threatened by its risks all the time. Having invalid or unverified requirements, being short of demanded resources, facing technologic difficulties, and getting affected from outside systems' risks are just examples of such threats. Fortunately, RAAM systems give us the opportunity to recognize and study the project risks and take proper actions to reduce them as much as possible. On the other hand, RAAM systems allow managers to choose the most reliable alternative among all existing possibilities that will enhance the reliability as well.

2. Second Level Goals

The second level of classification specifies more definite goals, as shown in Figure 3, and defines them as RAAM categories. These are the mid-term milestones, which lead the system to its first level goals:

2.1. Risk Identification

In order to increase the predictability of a system, different failure possibilities in any system phase should be known. The identification of such risks help the system managers gain a better understanding about the situation ahead of time and plan for them wisely. Risk identification is an intuitive task and needs deep knowledge and great experience. After participation in several similar projects one can understand the main risks of the same projects, which can be ignored easily in the first look. However asking suitable questions leads to better answers. By asking well-organized questions, risk identification systems can help risk officers pay attention to more details and find more project risks.

Any project consists of a set of components logically related to each other. Each component must wait until all former phases are done successfully and it is a prerequisite

for next phases in its turn. When there are several preconditions for a component, their relationship can be in the form of AND or OR. If several components are prerequisite for a phase in the AND form, the phase cannot be started until they are all ended. While in OR form, it is enough for starting the phase if only one of its preconditions is ended successfully. Figure 4 shows the difference between the two above forms.

Figure 4- AND/OR prerequisite relationships

A Contributing-Factor Diagram (CFD) or influence diagram is made at the early stage of risk management to identify main components of the project and their logical dependencies. These dependencies are so important in risk identification that one can say the more dependency we have, the more risk possibilities there are. Besides, the dependencies each project component has its own goals and criteria that can define component risks. The goals, criteria, and dependencies should be studied explicitly in order to define major risks related to each one.

There are many different aspects of risk threatening the components, like: legal, financial, personal, safety, environmental, technical, and so on. Therefore, the system should lead the users to think about all aspects with proper questions.

2.2. Measurement

After recognizing the risks in the project, now it is time for measuring and evaluating them. The final purpose of risk measurement is to rank the risks so that the system can

spend its resources optimally on the most important ones, which is a crucial step towards understanding the project deeply and predicting it properly. The ranking is done based on the risk value, which is the value related to the occurrence of the risk in any part of the project. This value is often – but not necessarily – calculated in term of money and depends on the following parameters:

I) Probability: The probability of risk occurrence is crucial in the importance degree of the risk. Risk probability has two definitions and so two different ways of calculation. The first method, Frequentism relays on the traditional definition of probability, which is the limiting frequency of happening. In this school of thought one can define an event's probability when she/he identifies a sample of independent, identically distributed observations of the phenomenon and counts the number of occurrences of the event. For frequentists, historical information of the organization or similar projects is the source of this sampling and risk engineers try to analyze a risk probability based on the frequency of its occurrence in similar situations. Classical statisticians and traditional risk managers usually use this method to estimate risk probabilities. The second way, Bayesian method defines probability as a degree of our belief on the credibility of the event. The supporting information in this school of thought includes not only statistical data and physical models, but also expert opinions. This method, in contrast to the former one, relays on subjective opinions that decrease the level of reliability but does not depend necessarily on historical data, which is not always available in many domains. [25]

II) Severity: The next important factor that is involved in risk ranking is the negative influence of the occurrence of the risk on the project. This negative impact is usually calculated in term of the monetary value. It can be estimated by experts or

derived from similar project history. The timeframe of the risk happening is also important because of the monetary value of time. There are different algorithms for the severity estimation.

III) Project Sensitivity: As we discussed in the definition of CFDs, any failure in a project component has also negative effects on the next phases, which depend on this component somehow. Fortunately with the aid of CFDs, we can easily find the mathematical formulas to calculate project sensitivity to the failure of any particular component. In fact a well-designed CFD is a blue print for generating equations that will form the basis for the risk assessment model. [21]

After the risk value is established for each risk, the system would prioritize the risks based on their value. Some systems prefer to categorize risks to fatal, major, minor, and ignorable categories at this stage.

2.3. Recording

In order to achieve a better planning and predictability in the next phases of current project as well as the next projects of similar natures, any risk identified or occurred at any stage of the system life cycle should be logged properly. The record would include name and nature of the risk, its occurrence date, risk owners and other stakeholders, management actions taken to handle it, its influences on the other risks in different levels, and its negative impacts. This information would be communicated among all stakeholders and would be confirmed by them.

Having a complete risk archive is helpful especially in those risk identification and measurement techniques that rely on former experiences and try to assess the current project's risks based on what has happened in similar projects. This archive would also

be useful in the organization evaluation processes, which are outside RAAM system but related to it.

2.4. Allocation

There are different types of resources in a project. Time, money, personnel, and equipment are some famous examples of project resources. There also are activities in any project that need those resources and therefore have conflicts among themselves. It is crucial for project success to plan for all activities and resources at the beginning and prevent any activity to use resources more than it needs. To do so, a critical task is to recognize all errands in the project and specify their requirements precisely so that managers in each section can plan to spend their share of the resources wisely.

Because having a task without enough resources is an important source of risk occurrence, resource allocation is closely related to RAAM system in terms of planning and increasing reliability. Risk managers need to be assured that all activities have proper amount of resources at proper times. They also need to know if there is any possibility to lose any of the resources or to have restricted availability to them.

Resource allocation is also important in RAAM systems from another point of view. Because RAAM activities are costly tasks, they need to have enough resources from all types as well as the other project phases. In fact, all decisions made in the next phases are highly dependent on the amount of resources available for risk reduction activities.

It is for the above reason that resource allocation is an important goal for designing RAAM systems.

2.5. Alternative Management

In any kind of project, there may be parallel activities that can be replaced without considerable effects on the whole system's performance. Take the small project of getting from home to the college as an example. One step of the project is to go up from the first floor in the engineering building. You have the choice to use the elevator or to go up by stairs. Each alternative may have its pros and cons. For example, going up the stairs may count as exercise and you do not have to wait for an empty elevator; while using the elevator is faster and easier. In different cases, you may have some criteria to consider too. In this example, you might be in a hurry and have limited time to spend getting up. Making decisions to select which alternative is an important part of planning and should be done based on different costs of each alternative. Risks are important parts of activity costs and should be considered accurately in the alternative management phase.

2.6. Risk Reduction

Once all risks in the project have been recognize and studied from various viewpoints, now it is time to take proper actions to reduce them. Risk reduction can be done in several ways. Risks with very low probability or severity or those needing a lot of resources to be reduced are usually ignored. It does not mean though that they are forgotten totally. The chance of changes in probability, severity, and sensitivity of a risk is always there and RAAM system needs to be aware of the latest changes in the system.

For some other risks, decreasing the probability of their occurrence or their negative impact on the system is advisable. Modifying the project component or injecting more

resources to a project part can do these decreases. Other risks may be completely avoided by some modifications in the project or resource schedules.

One important way to enhance the reliability of the project and reduce the risks is to transfer them to another organization. Buying insurance and hiring contractors are two popular ways in this category. Risk managers usually use risk transfer when they do not know much about the risk or because of monetary considerations.

Risk reduction regardless of its technique and methodology is the central part of the RAAM system. It has a direct relationship with increasing project reliability and should achieve enough attention in detail.

3. Conclusions

Setting the goals and missions of the system is the first step of the development of a pattern language. These core concepts categorize the subsystems and requirements of the system and help engineers to understand and therefore maintain the system more effectively. Because these concepts do not change during the system lifecycle, they are called Enduring Business Themes (EBTs) and are considered as the foundation of any system to build.

EBTs are all in the same level for big systems such as RAAM. They may be organized in more than one level to achieve more understandability. However, they answer the question: why build the system at all. For RAAM systems, predictability, planning, and reliability are the first level goals and identification, measurement, recording, allocation, alternative management, and risk reduction are the second level goals.

Chapter 3

How To Achieve The Goals

Any goal of a system will be achieved by using agents taking proper actions to fulfill that goal. These are workhorses that work together for a definite purpose and each have a certain role in the work process. As described in Chapter 2, the main missions of the system (also known as EBTs and Analysis Patterns) are the foundation of this study. This chapter describes the workhorses of each goal that actually do the business. These classes are called Business Objects or BOs.

BOs serve the EBTs to cover more general aspects of the problem and relate them to IOs, which represent application-dependent, tangible classes and are commonly replaced due to changes of requirements or application domain. Because the BOs have such an important role in firming the general design of the system, they are named Design Patterns. On the other hand, because of their strong relationship with EBTs, they are regarded as the properties of EBTs.

BOs also show the capabilities of the system and explicitly tell engineers how the purposes of the systems can be reached. Design patterns remain externally stable but internally adaptable during the system's lifetime. This is because of their internal dependency on the current technology that requires some minor modifications when new techniques are used. However, the design should be stable enough to hide all these minor adaptations from any outside client so they will find the BOs externally stable. Here is a list of characteristics that help one declare proper BOs of a system:

- BOs are stable patterns that help the system achieve its goals. They are workhorses to actually do the required business to fulfill a general EBT.
- BOs are general enough to be applied in any application area with the same core concept. So, limited application-oriented concepts cannot be defined as BOs.
- BOs are neither tangible nor replaceable in any system. Some semi tangible items -mostly human roles- are also counted as BOs.
- Proper BOs address the question: “What does a system need to achieve a particular goal?”
- BOs show the result of preliminary design phase. They include the main workhorses of the system so we call them also design patterns.
- BOs, like other parts of PL, will be our system’s classes. They should have unique responsibilities and related properties in their own. If a candidate does not contain obvious data or functionality, it probably is not a BO.
- Unlike universal EBTs with exactly the same concept in any application, BOs may have several different instances in various applications. For example, there might be only one obvious meaning for EBT Measurement, but there are so many different metrics to be served as a BO for this EBT. In order to emphasize this difference, word “Any” is usually added to the beginning of BO names, as you will see later in this pattern language.

1. Explicit Capabilities

Applying the above-stated phase on the categories found in the last chapter, here are a few properties for each EBT.

Related EBT	Property
Risk Identification	AnyProject AnyTask AnyMilestone AnyResource AnyDependency AnyRisk AnyRiskIdentificationMethod
Measurement	AnyMetric AnyProbability AnySeverity AnySensitivity
Documentation	AnyLog AnyEntry
Alternative Management	AnyAlternative AnyCriterion AnyCost
RiskReduction	AnyAcceptation AnyAvoidance AnyTransferring AnyMitigation
Resource Allocation	AnyResource AnySchedule

Table 3- BOs in RAAM system

This part of the chapter describes the properties for the most prevalent BOs. Each section includes a general definition of the term as well as its connectivity to our goals and missions.

1.1.AnyProject

A project is a discrete collection of tasks, which are required to fulfill a specific set of related goals, and/or resolve a specific development problem. The definition of the

project along with the determination of its goals and resources is a crucial prerequisite for risk identification, which was one of the RAAM second level goals. Projects are usually defined with the introduction of necessary tasks to be done as well as the main missions that and needs to be reached. These and more related concepts are also captured in the next design patterns.

Figure 5 shows a pattern structure for AnyProject Stable Design Pattern. A template to explain this capability of the system in detail is located in appendix A: Pattern catalog.

Figure 5- Stable design pattern for AnyProject

1.2.AnyTask

A task is: “a usually assigned piece of work often to be finished within a certain time”, as it is defined in the Webster Dictionary. It is supposed to be done using particular amounts of resources to reach a set of milestones. Tasks are the building blocks of a bigger project so identification of tasks and deep study of their relationships is a crucial job in order to identify major risks in the project.

1.3.AnyMilestone

Any task has at least one milestone which is its final mission described clearly. Milestones should be expressed as zero-one terms so one can easily evaluate if a milestone is reached in any given time. The definition of milestones is very important in any project both to identify the project itself and to verify the project-finished declarations at last. Milestones also clarify the goals of each task, so they provide useful information for risk identification activities.

1.4.AnyResource

Some encyclopedias define a resource as “a source of aid or support that may be drawn upon when needed.” Any project or task needs some resources such as time, money, people, and equipment and one important cause of risk is to have a task with insufficient or insecure resources. So managers should be sure about availability and reliability of different kinds of resource for each task and should monitor the assigned risks to ensure that resources are in hand when needed.

There is a stable design pattern developed to analyze and design AnyResource in more depth. After describing the problem fully and precisely, the pattern template offers a design diagram as a stable solution for the problem. Figure 6 shows the pattern structure of AnyResource design pattern as it is illustrated in reference [26].

1.5.AnyDependency

Wikipedia, an online incyclopedia, offers the following explanation for dependency:

In project management, a dependency is a link amongst a project's terminal elements. There are four kinds of dependencies with respect to ordering terminal elements (in order of decreasing frequency of use):

1. Finish to start (FS) : A *FS* B = B doesn't start before A is finished.
2. Finish to finish (FF) : A *FF* B = B doesn't finish before A is finished.
3. Start to finish (SF) : A *SF* B = B doesn't finish before A starts.
4. Start to start (SS) : A *SS* B = B doesn't start before A starts.

Figure 7- Four forms of dependency between timed elements -Resource:
www.wikipedia.com

There are three kinds of dependencies with respect to the reason for the existence of dependency:

1. Causal (logical) like *“It is impossible to edit a text before it is written”* or *“It is illogical to pour concrete before you dig the foundations.”*
2. Resource constraints like *“It is logically possible to paint four walls in a room simultaneously but there is only one painter.”*
3. Discretionary like *“I want to paint the living room before painting the dining room, although I could do it the other way round, too.”*

Understanding these dependencies among the project tasks is helpful for any system to determine how any failure in one part of project affects all other parts and how much the overall negative impact of such a failure on the project is.

1.6. AnyRisk

Risk is something that might happen and have a negative impact on a project. In other words, it is the potential harm in the future that may arise from some actions in the present. Any risk is a central BO in this system that has important roles on almost all EBTs of the system. Risks will be identified and measured based on their main characteristics. Information on risks also needs to be documented for later uses in the both current and future projects. The next step is to select between existing alternatives, if there are any, regarding their costs and to plan for risk reduction. Then, some resources will be allocate to each risk based on the chosen risk reduction plan.

Appendix A: “Patterns Catalog” includes a full pattern template describing the AnyRisk pattern. It resolves all the problems existing with the current solutions offered for risk problem after a comprehensive study about risk problem, context, and forces. Figure 8 shows the pattern structure, developed for this stable design pattern.

Figure 8 -AnyRisk stable design pattern

1.7.AnyRiskIdentificationMethod

Risk identification is not a flat task. It is involved in so many different levels of knowledge and documented experience. There are many informal methods used to assess risk as well as a few formal methods such as value at risk used in some applications. Having a separate pattern for risk identification will isolate all issues and varieties of these methods and open up the room for ongoing improvement in this area.

1.8.AnyMetric

A metric is a measurable representation for an immeasurable thing. For example, one cannot directly measure a system's complexity but can use the number of components as a representative of the concept of complexity and measure it instead. Using metrics is unavoidable in measurement especially when it comes to uncountable concepts. However, there are debates about descriptive value of metrics and best methods to choose the right ones.

1.9. AnyProbability

Webster Dictionary describes probability, as “a measure of how likely it is that some event will occur.” Measuring the probability of occurrence of risks is crucial to rank them based on their importance and to take proper decisions in order to spend the available resources optimally to enhance the project’s reliability as much as possible. The probability of failure for a task is a combination of the probability of all possible risks that would happen both in this step and in all steps that this step depends on.

1.10. AnySeverity

In order to prioritize risks, managers need to know how severe each of them is and what direct and indirect impacts on different parts of the project their occurrence will have. Because there are tremendous natural differences among all possible severities of various risks, RAAM systems usually evaluate the monetary value of the severity, which is the required money to fix the damage or its resulted loss of profit.

1.11. AnySensitivity

Risks affect the project in several ways if they occur. They cause various types of damages to properties and targets of the project, which would be studied and calculated while inspecting the risk’s severity. However, different risks in a project are not totally independent. They have their internal dependencies, so that occurrence of one risk can influence probability or severity of others. Take a farm as an example. Having a drought may cause a lot of severity for agricultural products. Moreover, it will increase the risk of fire accidents by decreasing the humidity. It also may increase the risk of particular diseases for planets and animals. However, fire accidents or diseases have no reverse influence on the risk of drought.

This simple example shows that there are always some risks that are crucial to be mitigated not necessarily because of their high probability or severity but for their negative influences on other risks. This type of importance relates to the sensitivity of the project to the risk and it is represented by the amount of increase in the overall risk value of the project when the probability of this particular risk goes up a few percent points.

1.12. AnyLog

AnyLog is the selected name for any kind of formatted log to contain any kind of information records to be documented. Database management methodologies offer tons of ways to organize entries. However, with all possible differences, the concept of entry organization in a logical way to optimize the required time and effort for data storing and retrieving is common in all of them.

As a complete pattern template, reference [27] provides an accurate analysis for AnyLog concept and derives a conceptually stable design as the solution. As an advantage of using SSM methodology this stable pattern is also reusable in RAAM case without any modifications. The offered design of AnyLog in this reference is like what comes in Figure 9:

Figure 9 - AnyLog stable design pattern – resource: reference [27]

1.13. AnyEntry

Documentation is all about storing some information records that will be used later. These records are called entries and therefore Entry would be a very important property of EBT: documentation. In RAAM systems, the most vital information is the risk-related one, which would be used as guidance in later risk identification phases.

1.14. AnySchedule

Available resources including personnel, money, time, equipment and more, are crucial assets in any kind of project. It is necessary to determine project resources. Moreover, resources need to be assigned to different project components. This managerial plan for resource allocation is called scheduling, which plans for more

resources than time only. It is important to determine project schedule in this PL not only because risk management needs its own share of resources but also for the fact that defining the project schedule is a central task during the risk allocation phase.

1.15. AnyAlternative

Some projects require more than one set of tasks to achieve a particular milestone. These parallel series of tasks, which can resolve the problem independently, are called alternatives for that milestone. In this case, all alternatives should have the acceptable performance to guarantee achievement of main goals and missions.

Figure 10 -Tasks T1,T2,T3,T4 and tasks T1,T5,T6,T7: two alternatives for the same target

Each alternative though, has its own characteristic such as certain resources required to do the job and overall risk of the path. These characteristics will be the base for managerial decision about which series of tasks we should select to maximize the project quality factors such as reliability and minimize the project cost.

1.16. AnyCost

Each alternative requires certain amount of resources such as money, time, personnel, and equipment. Choosing the cheapest alternative, managers try to minimize the cost of achieving the milestone in terms of the above-mentioned resources. Nevertheless, there is a hidden cost for the project, which is the risk impact cost. In most projects with risky situations, managers especially prefer to choose the alternative with

the lowest risks, i.e. they consider risk impact as an important part of each alternative cost.

1.17. AnyCriterion

During alternative management, we select one of potential alternatives because it has the minimum cost to the project among all acceptable candidates. However, in any alternative management there may be some extra criteria for acceptable choices that should be considered carefully. These criteria can be some extra restrictions on resources or some demanded quality factors. In both cases, we need to measure the demanded factor in all potential alternatives and make sure to select among those with acceptable qualities.

1.18. AnyAcceptation

Once risks have been identified and ranked, proper actions must be taken to reduce them. This would be possible in different ways depending on the probability, severity, and sensitivity level of each risk. For those risks with very low severity and relatively low sensitivity, a lot of money is needed to reduce a risk, which is not that important even though it happens. So usually managers prefer to accept the risk without further risk reduction actions. This does not mean that they do not need to keep track of the accepted risks. They still should monitor such risks both to verify our measurement methods and to ensure that the risk characteristics do not change over time. It is always possible for risks to be influenced by some event and move from a category to another.

1.19. AnyAvoidance

Another way to deal with a risk is to try to avoid the risky situation. Take the risk of mistakenly changing important files in a computer system as an example. If the

probability and severity are relatively high, we may decide to prevent people from writing access to the files. That way we avoid the risk from occurrence. However, avoidance strategies may pose more costs to the system. In this case, updating such files would be very difficult and people would need to replace them more often.

1.20. AnyTransferring

Transferring the risk to another organizations or individuals is another popular way to deal with risks. There are situations that the managers decide to hire some body to do the risky job to reduce the overall costs. Buying insurance policies is another example of risk transfer. The common base for all these solutions is to avoid the risk not by eliminating the risky path but by paying extra and constant amount of money to avoid one big loss.

1.21. AnyMitigation

Finally, risk mitigation is a good way to reduce the risks. It includes any way to decrease the probability or severity of a risk. Remember the example of file deleting problems. Some times a secure access control system such as double-checking messages after the delete command can reduce the probability of mistakes. Another way could be backup creating that would reduce the negative impact of mistakes. Risk mitigation method is worth when risk sensitivity is fairly high.

2. Common Capabilities

There are two more important BOs that are common among all EBTs. They are recognized widely to be a part of any pattern diagram even if they are not mentioned explicitly for simplicity improvement. Here are these common design patterns and a small explanation for them:

2.1. AnyParty

AnyParty is a required aspect of any mission because they always need some human assets to participate, manage, evaluate, or monitor the processes. There is no completely automatic system that works without any need for human participation. Even if such a system existed, personnel or organizations design, finance, use, or maintain the system. AnyParty design pattern has been described in more depth in reference [20] and Figure 11 shows the structure that has been offered there as the final solution:

Figure 11 - AnyParty stable design pattern – resource: reference [20]

In RAAM pattern language AnyParty represents active participants in the system such as project managers, risk engineers, end users, project owners, project employees, etc. These parties have specific roles in the application that would be specified during the resource allocation stage.

2.2. AnyMedia

AnyMedia is another widely utilized BO in most EBTs. It is a surrounding environment that facilitates the communication among different agents. Because communication is an essential part of most systems we usually have medias as design patterns in our systems as well. AnyMedia usually transfers messages and materials

among parties and makes them applicable for each other. Internet, computer systems, phones, and bureaucratic channels are most popular examples for AnyMedia pattern that can also be used in RAAM systems. Figure 12 is the solution that reference [20] provided after a comprehensive description of AnyMedia design pattern:

Figure 12 - AnyMedia design pattern – resource: reference [20]

3. Implicit Capabilities

A big system like RAAM cannot be designed in just one layer without a troubling amount of complexity. In order to avoid such a crowded and complicated design, a multi-layer design diagram is created with several depth levels. In this diagram, each level may contain one or more BO-patterns that represent another layer of design diagrams. The hierarchy goes on until we reach atomic classes in the last levels. The system contains capabilities other than the above BOs, which would appear inside the design models for pattern BOs such as *AnyRiskIdentificationMethod*, *AnyProbability*,

and AnySeverity. These other capabilities are implicit unlike the explicit capabilities that were previously described.

Because the number of implicit capabilities is very huge in the RAAM system, this thesis will only describe a few of them. Implicit patterns will be discussed as part of pattern templates for explicit capabilities. Some potential implicit BOs are briefly described in this section. See references [24, 29] for more details.

3.1. Fault Tree

Fault tree is a tool for predicting system's reliability and availability parameters. It is a way to organize all possible risks in a hierarchical top-down structure that is fed by all risks along with their probabilities and severities. Then the overall failure's probability and severity will be calculated with Fault Tree Analysis (FTA) approaches that involve a graphical enumeration and several analysis techniques.

3.2. Cut Sets

Cut-set analysis, which identifies a minimal set of activities in a fault tree, event sequence diagram, or scenario, is one of the major ways to find out the weakest links in the project. It shows the critical set of project activity failures or risks that can fail the whole system if all occur simultaneously. Several mathematical algorithms have been developed in graph theory to find the minimum cut set in a graph.

3.3. Mission Timeline

One of the most important roots of risky situations is project tasks' conflict for available resources. In order to identify and eliminate such conflicts, the mission timeline diagrams are developed to show all project tasks with their allocated resources and the time period that each task is in control of the resource.

3.4. Reliability Block Diagram

Reliability Block Diagram (RBD) is a technique to calculate the unavailability period and failure frequency of systems. It shows each project task as a separate block and relates the blocks based on existing dependencies among the tasks. It uses some approximate methods that would calculate system reliability parameters in a reasonable time especially when the diagram is large and complicated. Engineers can combine simulations or analytical methods to prepare extremely useful data about overall reliability of a system or subsystem, when they use RBDs.

3.5. Event Sequence Diagram

Event Sequence Diagram (ESD) is a powerful tool to assess and measure the risks. An ESD is a visual illustration for a modeled project that shows a set of possible events that can occur sequentially and their overall impact on the system. Each sequence of occurrences and non-occurrences of key events makes a different scenario in an ESD that should be measured as a possible outcome of the system. Each scenario starts from a beginning state and eventually leads to an end-state, which specifies the result of the particular scenario. A sensitivity analysis process, allows the engineer to study any changes to probability and severity of risks in ESD pivotal events.

3.6. Component diagram

Component diagram shows different components of a project. It presents each component with its goals and available resources and emphasizes on the dependencies among different components. It can be used to assess and measure the risks and to do what-if analysis before taking the decision.

3.7. Binary Decision Tree

Binary Decision Tree (BDT) is an effective technique for system and component evaluation. It is also useful for alternative management especially when the events are internally related so that any choice would affect the other components of the project. It starts with identifying the initial options and follows the subsequences of each sequentially to the last level. Then it calculates the value of each option based on its results.

4. Conclusions

After setting the goals of the system, engineers need to study related knowledge areas and to think about the EMTs conceptually to identify workhorses the system needs to fulfill its missions. These capabilities of the system work together to actualize our final goals system BOs and define them in the second phase of pattern language development. These BOs are categorized into three types of explicit, implicit and common BOs and identify and describe several patterns in each category. Several pattern structures have been briefly described and a few pattern templates have been addressed for more information.

Chapter 4

Architectures in Demand

There are recurring problems in any scientific domain. These are issues that appear in different ways with the same conceptual meanings. A pattern is a general solution for those recurring problems resolving them in a more abstract level without addressing domain dependent issues. So patterns are supposed to be highly reusable within their area of research. “A pattern is the abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts.”[38]

Software patterns have been introduced and justified widely since the wide acceptance of the book “Design Patterns: Elements of Reusable Object-Oriented Software”[30]. They have been a hot topic for scientific research in different domains ranging from organizations and processes to teaching and architecture. Currently, the software community uses patterns mainly for software analysis, design, and architecture as well as software development processes and organizations. Other important books that introduce popular patterns are listed as references [37, 38, 39].

A good definition of software patterns came in reference [31]: “A pattern is a named nugget of insight which conveys the essence of a proven solution to a recurring problem within a certain context amidst competing concerns.” The author also adds later: “a pattern does more than just identify a solution; it also explains why the solution is needed!” This means that we need more explanations about different aspects of a pattern to convince people that it is a right solution for their specific problems and to help them use it correctly and effectively in their applications.

There are different templates that specify which information items should be prepared to describe patterns. Using such a standard is a good idea because they make pattern developers describe their patterns as completely and precisely as possible. They also provide a standard terminology that would help any engineer understand and compare developed patterns and finally select the best one to fit a particular application. Having standard templates such as the pattern template introduced in reference [20] to document some related patterns in Appendix A are also useful to build big archives for reusable patterns that increase availability and comparability for patterns.

Also known as an architecture, a framework is a system of patterns working together to support a family of similar applications as a general backbone design [32, 33, 34]. It is “a set of cooperating classes that make up a reusable design for a specific class of software. A framework provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations. A developer customizes a framework to a particular application by sub-classing and composing instances of framework classes.”[30] In other words, “A software framework is a reusable mini-architecture that provides the generic structure and behavior for a family of software abstractions, along with a context of memos/metaphors which specifies their collaboration and use within a given domain.” [39] The difference between a framework and an ordinary programming library is that a framework also controls its relationship with its clients. In order to use a framework, engineers usually implement some hooks [35] to customize the general architecture to their problem in the form of a few classes. The framework does the rest of the work, invoking necessary client callbacks or methods at the appropriate time and place. Architectures usually use several analysis and design

patterns that are smaller and more abstract. Risk Assessment unit which is a gathering of EBTs: Risk Identification, Measurement, and Documentation along with the related BOs is a good example of a set of EBT/BOs that make a popular framework to be used in many different instances. See Chapter 5 for full descriptions of the process of instantiation. The story, however, does not end here. Architectures related to the same area have their communication and contractions. They need each other's data and functionality. A group of interrelated frameworks form a pattern language, which documents all related patterns to a particular area of problems. "A pattern language is a structured collection of patterns built on each other to transform needs and constraints into an architecture", Reference [39] states. It continues then: "Unlike a mere pattern compilation or catalog, a pattern language includes rules and guidelines which explain how and when to apply its patterns to solve a problem which is larger than any individual pattern can solve." These rules and guidelines demonstrate how to construct arrangements to satisfy the needs make the necessary semantic for a system of patterns to be called a language. A knowledge map is a diagram to present a pattern language with all architectures in it. It includes many directions to which different patterns are related.

Figure 13 – Knowledge map, architectures and stable patterns

However the development sequence of stable patterns, architectures, and knowledge maps is not the same. Usually designers come up with proper stable patterns after they

study the problem domain. Then they develop a knowledge map including all related stable patterns and their relationships. The next step is to suggest different architectures that are already implanted in the knowledge map. Any stable pattern in the map pursued from any potential direction forms a framework made of some related patterns for a particular application family. There are several possibilities for building different architectures by selecting any given set of related patterns and any architecture can potentially turn into various applications by adding proper hooks [35]. See figure 14:

Figure 14 – The development sequence for stable patterns, knowledge map, and architectures

Because the reason for providing the map is just to show the conceptual relationships among the patterns, specifying the quantity or type of relationships in this level is not necessary. This is a good way to maintain simplicity and understandability of the map, which are two critical quality factors for such a dense summary of the pattern language. Pattern structures, instead would include enough descriptions of the nature of relationships and the quantity factors.

Figure15 shows the map of any RAAM system. It includes all formerly introduced stable patterns and their relationships. In order to maintain the simplicity of the map, common BOs such as AnyParty and AnyMedia are omitted from the map. However, it is evident that they participate in any architecture and scenario. The map shows some patterns are marked with the sign:✱ This means that a more detailed analysis or design pattern template has been developed for the marked pattern. The blue marks mean that

the pattern template is prepared as part of this dissertation and can be found in Appendix A, while the red ones show that the pattern is developed elsewhere with the same technique. As an important advantage of software stability, patterns are developed in a very general way so that they can be reused in any context including RAAM systems. Chapters 2 and 3 briefly include the solution for core problems of developed patterns. References [20,26,27] provide a detailed description.

Figure 15 - Map for RAAM systems

There are other maps in the system too. They show detailed information about other related concerns of the system like its deployment and development considerations. The following chapters explain these two important aspects of a system after the establishment of its operational aspects. Because quality factors and deployment considerations are counted as important goals of the system, separate EBTs are defined to represent these concepts. Each EBT as usual would have a set of BOs as the workhorses that actually do the job. Complete maps for development and deployment concepts illustrate the relationships among those EBT/BOs. Development and deployment concepts are related to the operating ones, so it seems natural to gather all patterns in one big map. However, keeping them separate makes them simpler and easier to understand.

Conclusions

After determination of stable patterns, designers need to organize them in a knowledge map showing all stable patterns related to a domain and the relationships among them. Such a conceptual map helps engineers understand the problem better and extract proper architectures for relative problem families by following any direction in the map starting from a given pattern.

Knowledge maps are supposed to be simple enough for high understandability so detailed descriptions of patterns and their relationships are left to separate pattern structures.

A knowledge map that conceptually represents RAAM systems has been provided in the chapter.

Chapter 5

From Design To Implementation

There is an information gap between the design and the implementation phase. This is because the phases have different characteristics and priorities. Designers and code engineers do not necessarily share common language, terminology, knowledge, skills, and experiences. Usually people in the design phase make some assumptions that are not trivial for implementation engineers. There are usually communication problems between the engineers in the two phases even when they work at the same organization. Moreover, usually people buy a pattern language for reuse and have restricted access to designers. Finally, most engineers involved in the implementation phase are not that familiar with design activities and do not necessarily understand the processes of converting a general design to a particular application.

To fill the gap between design and implementation phases, it is very important to log enough information in details in formal documents. The importance of this is evident because of the common misunderstandings during implementation. See figure 16:

Figure 16- Communication between design and implementation teams

This thesis has introduced patterns that represent main concepts of the area and shown their relationships. The next step of providing a pattern language is to identify the main processes in the system and prepare enough descriptions to support further reuses in the form of development patterns. Development patterns are algorithms and processes that are common in all applications in the area. It includes not only the algorithms and processes that show how to develop an application based on the pattern language but also the main technical algorithms in risk analysis, assessment, and management. In fact the latter is more important than the first. These patterns teach users how to do the processes and will not be present in the final design as separate classes. Development patterns are powerful tools for documenting the main procedures in the system precisely.

In RAAM, there are some important groups of process patterns to be explained:

- There are so many different techniques in RAAM. Different techniques of risk identification, various algorithms of risk measurement, and diverse methods of selecting a proper risk reduction plan are some important examples of such variety. These techniques may appear as separate patterns in the second level designs - as mentioned in the discussion of implicit patterns in Chapter 4 - or may be utilized as some operations inside other patterns. In the first case, stable designs are developed for them just like other patterns in the PL so that they will have completed templates as a mean of description and documentation. In the second case, process patterns are good ways to document these methods. Process patterns will make the job of developers much easier in the latter situation.

- For any identified pattern, designers need to describe the detailed design precisely. Of course, there is a chance to reuse some patterns developed by other designers. If this is the case, they need to completely address the pattern and evaluate it based on forces, advantages, restrictions, and different quality factors. References [20,26,27] show some patterns that can be reused in RAAM systems. They were briefly described in the last chapter, but the process of choosing the right pattern and evaluating it properly shall be considered and explained fully as a separate development pattern.

- For new patterns, various aspects of the design need to be explained completely and clearly so that anybody can understand it easily. In order to have a standard for pattern documentation, a template has been developed [20]. This pattern template consists of different sections to cover the problem itself, the solution offered in the pattern, and an overall evaluation of the pattern. As another important part, this pattern template requires a separate section to show the applicability of the pattern in different examples. This part presents the reusability value of the pattern in various usecases demanded in any given application area. The process of filling this template and examining the result for the quality factors should also be defined in separate development patterns.

- The process of identification of IOs and plugging them into the pattern language is a key process for a safe reuse. This includes the introduction of hooks and the how-to-use guidelines, which has a close relationship with the concept of scalability. Hook is an information package to determine the required stages to identify and locate one or more IOs in a developed architecture in order to customize it for a particular application. “A hook may involve something as simple as inheriting from an existing class in the

framework, or as complex as adapting the interaction of a large number of classes within the framework.”[20]

Reference [20] introduces the concept of hook and offers a standard template to describe and document all aspects of a hook for later reuses.

Conclusions

There are four groups of important processes that show the future work that one has to do in order to reuse a pattern language for a particular application:

- 1- The technical algorithms used in the implicit patterns (see Chapter 3)
- 2- The process of selection and reuse for the patterns that have been already developed by other engineers
- 3- The process of pattern development for demanded patterns, which are not on the shelves
- 4- The process of instantiation and adding proper hooks to the architectures for application development

All these process patterns should be considered parts of the pattern language as well as the patterns that are introduced in former chapters. Development of such patterns is another step of PL development.

Chapter 6

Setting The Main Direction For Quality Assurance Activities

Stability based pattern language strongly recommends engineers to do validation and verification tests at the end of each phase to make sure that the deliverable products of each one is provided exactly as they were supposed to be. It would help them find the flaws as soon as possible, which would result in a lot of time and money saving in quality assurance process. It also would solve a lot of problems regarding test design and deliverable specifications because it forces any production to have its own clear definition and quality factors right at the beginning of production. Another benefit of this suggestion is to make it easy to find and track bugs and prevent situations where some flaw has been identified but nobody to take responsibility. Figures 17 and 18 show this suggestion:

Figure 17- Project lifecycle with one testing step at the end

Figure 18- Project lifecycle with several testing steps after each phase

However there should be a separate phase to plan and design the main directions for quality assurance activities. In this phase we intend to emphasize more on the quality

factors for the project as a whole and the recommended ways to obtain them. Having these main quality factors, QA engineers will design detailed QA plans during the testing phase for each deliverable product.

In order to identify deployment patterns, designers should consider any special consideration that may affect their preliminary design. These are very important quality demands, sometimes known as non-functional requirements. Scalability, reusability, adoptability, stability, and cost effectiveness are some examples of these issues. Then they should provide enough description in the form of more EBTs and BOs to show how these quality factors are achieved and kept in later applications.

1. Reusability

RAAM is a widely demanded system with so many different application instances in various areas. From agricultural programs to space shuttle missions, almost any kind of project faces risks and therefore is a potential user of this pattern language. Thus a crucial characteristic of RAAM should be its reusability to cover all these differences.

In order to minimize the cost of optimization of a design to different applications, domain-dependent parts of the design should be separated from the abstract and conceptual ones, which is exactly what has been suggested in software stability guidelines. Having tangible items that are relevant to a specific area as separate classes (called IOs) is a very wise way of separating common knowledge of the area from replaceable hooks. However, to achieve the best reusability level designers need to design the hooks carefully and to cover as much data and functionality as possible in the core designs so the IOs are left with the minimum of content. That way they minimize the effort required for reusing the pattern language in any application area.

2. Scalability

Software systems are always in change. New requirements are introduced to the system all the time and current requirements are modified constantly. This is the reason why modern software systems suffer from a huge amount of maintenance cost and a short period of time between successive reengineered versions. This is especially true for a hot research area like RAAM that is currently developing. So every RAAM system design should consider the scalability as one important quality factor to keep effectiveness of the system after new demands and minimize modification costs. This would include two different kind of scalability:

I) Vertical Scalability

Vertical scalability is related to having the current system extended by adding more goals and missions. Suppose that one needs a risk management system to report to certain authorities after the occurrence of each risk. This is a new requirement that normally is not considered for a RAAM system. However, the system should have the required vertical scalability level to accept this new mission as another EBT and make the necessary relationships between this EBT and others.

In order to achieve such a quality, the pattern language needs to have accurate process patterns to cover all aspects of definition and integration of a new EBT to the system. These process patterns that describe how to scale the PL vertically would be described fully during the development phase.

II) Horizontal Scalability

Horizontal scalability is considered when a system needs to utilize other tools and techniques to achieve its current goals. As an instance, take an organization with a high

demand of logging different documents. Suppose that this system is used to keep the records on paper files in a huge folder archive. However, as a result of new database technologies, the managers decide to use a new computerized system that keeps the information digitally on CDs. This scenario happened during the last decades again and again and caused a tremendous amount of resources wasted to adopt new technology.

Horizontal scalability is one of the benefits of utilization of stability based pattern languages, since it uses IOs as tangible items of the problem domain that would be easily added, removed, or replaced with no considerable effort and cost. The concept of a hook is a crucial concept in this process. “A hook is a means of representing knowledge about the places in a framework that can or should be changed or augmented by application developers to produce an application from the framework” [20] It gathers the necessary information for engineers to learn how to change a framework to be scalable and/or reusable for a new application. “A hook may involve something as simple as inheriting from an existing class in the framework, or as complex as adapting the interaction of a large number of classes within the framework.”[20] Each hook description consists of a name for the hook, its type, the requirement(s) that will be satisfied using the hook, the framework area affected by the hook, the other hooks required to be used by this hook, the hooks participants and the main section of the hook that outlines the changes to the interfaces, associations, and control flow amongst the components given in the participants section. It also offers the constraints that limit the usage of the hook as well as any additional descriptions. For more information on hooks see the reference [20].

3. Understandability

Understandability is another important quality factor for any RAAM pattern language. This is because unlike most pattern languages, people who are not necessarily computer experts would reuse it in different fields. They need to easily understand the map as a whole, and properly apply the general solution to their problem statement. Having enough descriptions in all different parts as well as offering clear examples for simple applications can help a lot maintaining this quality factor in a RAAM pattern language. Complete documentation of every pattern in standard templates besides having simple designs along with informative and well-chosen names will hopefully solve the problems of this aspect of understandability.

Understandability has another facet in RAAM systems, which is as important as the former one. It is because RAAM systems have a large number of participants that need to communicate effectively and precisely. Designers can appreciate the importance of unambiguous communication when they consider most messages in the system are qualitative and crucial at the same time. With the absence of quantitative figures, misunderstandings can easily happen and because they talk about a strategic system big loss can result. To improve understandability among parties, the system should emphasize on a dense communication using standard formats and predefined terminology that would decrease chances of ambiguity.

4. Traceability

Risks have their own lifecycles from the moment they are identified until the end of the project. Their characteristics -especially probability- may be influenced by many factors in the project including other risks. On the other hand, there are different parties

involved in RAAM activities that need to effectively communicate their information among themselves and to a central documentation archive. These are the reasons why tracability is so important in RAAM systems that it can be declared as another quality factor for this family of systems.

Because tracability of risks is highly related to the communication opportunities among the stakeholders, it has a strong relationship to understandability, the quality factor discussed in the former section. In fact, to make all parties have a proper understanding of what is happening to each potential risk, the system needs to have them communicate effectively and as a result of such a successful communication, parties will be able to track the risks accurately during the project lifetime.

The above mentioned quality assurance aspects of RAAM systems can be summarized in the Figure 19:

Figure 19- Deployment map for RAAM systems

Reusability, Scalability, Understandability, and Tracability can be treated as EBTs and expanded in separate pattern templates. This way we can analyze these software requirements fully and offer precise solution for them. Engineers that participate in software development should read these pattern templates and consider the solution points all over the system's lifecycle.

5. Conclusions

One of the most important parts of stability based pattern language development is to set a series of goals for quality assuring activities and describe the way to reach them in separate patterns. This chapter introduced Reusability, Scalability, Tracability, and Understandability as the main quality factors for RAAM systems and explained them cohesively. These attributes were organized in a quality map that visually shows the quality factors' interrelationships. A crucial step to be taken in the future would be to provide separate pattern templates for each concept and document it for later references.

Chapter 7

Wrapping It Up

Risk Assessment, Analysis, and Management (RAAM) systems cover different aspects of project management. They are important tools to improve the project predictability, planning, and reliability, so they save valuable resources such as time, money, and equipment and maximize the project profits. RAAM systems identify the major risks of the project. Then they measure them and prioritize the risks based on the risk values, which are derived from risk probability, impact, and sensitivity. The next step of RAAM is to record each risk with all its related attributes and allocate required resources to the risks so that later risk management activities can be done properly. As the next stage RAAM systems should select the best among potential alternatives based on all defined criteria and costs including risk-related ones. Finally, the proper actions are taken to reduce the risk. Acceptation, avoidance, mitigation, and transferring are different types of risk reduction activities and their choice would be based on the project's characteristics. The final goal of all these activities is to maximize the return of project investment.

Gathering the knowledge of this area and identifying its core concepts is a crucial task. It would help involved parties to understand RAAM activities and to determine a well-defined process to be used clearly. It also will serve as a backbone design for any application with similar missions and goals. This is why providing a pattern language for Risk Assessment, Analysis, and Management was taken care of as the main theme of this dissertation in the former chapters.

1. Stability-Based Pattern Language

The methodology used in this thesis is stability based pattern language. It starts with identifying the missions of the project. Then designers should define the workhorses and system capabilities that work together to fulfill that goal. The next step would be setting the quality factors for the system so they would be considered in all steps of system development. Finally, the process patterns for development phase should be determined. These development patterns are either the main algorithms commonly used in the system or the process of preparing the patterns for implementation. Then the relationships among all defined patterns are represented in a knowledge map.

There are many benefits with the concept of stability based pattern language, which I will mention some important instances here. It is appropriate after we introduced and used the main guidelines during the former chapters and observed the benefits actually.

I) Stability based pattern language is applicable to any project in any area of knowledge. This methodology can be used to understand all problems in the life and prepare appropriate solutions for them.

II) At the same time, stability based PL guidelines are developed considering the main needs of the software industry and business affairs as two important client of pattern languages. Anybody who is familiar with the terminology and concepts of these two areas would find a lot of links and references to them while building a PL.

III) Unlike the other ways to develop a pattern language, stability-based pattern language provides a step-by-step approach for problem analysis and solution design, which is not based on artistic innovations but follows a set of rational roles.

Everybody can learn and use the well-defined heuristics and they will lead him/her to the success.

IV) The solution designed by the methodology is very general so it can be reused across different domains in any application with the same missions and goals. It is highly scalable both to cover new missions and to utilize new tools and method to achieve an existing one.

V) Thanks to the clear map and complete templates, the result of the stability based pattern language is highly understandable. It does not take much experiences and education to reuse them in particular applications.

On the other hand, there were some difficulties in the application of stability-based pattern language. The main point is the more number of classes that would increase the initial cost of pattern language development. However, the above benefits along with the minimized cost of system maintenance justify such extra cost. It is especially recommended when the system has a high level of reusability in different application. This methodology is well suited for big projects that would return the initial cost when it comes to scalability, reusability, and maintainability.

Another obstacle of the utilization of stability-based pattern language is to cover the complete information about the common concepts and about their different usages in different areas of knowledge. Unfortunately, there are so little references that gather such information maybe because there was not enough attention to the matter. One important step is to work on various aspects of common concepts.

2. Contributions

Risk Assessment, Analysis, and Management is a well-known practice for most project types. It deals with identification, measurement, prioritization, documentation, and reduction of project risks and making related decisions like resource allocation and alternative management. These activities are demanded in most projects because they enhance project predictability and reliability, as well as they help managers to design more qualified plans.

This thesis provided solutions to the problems existing with the contemporary patterns for RAAM and their implementations by developing a stability based pattern language. The developed PL can be used in any domain and encompasses all the essential features to get the system work. It ensures that the pattern is flexible enough to be applied to many domains and remains stable even after addressing new requirements and design changes.

The pattern language development for RAAM system was a challenging task and it involved a substantial number of patterns as you have seen in categories of analysis, design, architectural, and process patterns. The main part of the work was to identify these main patterns within the area. A comprehensive knowledge map was developed to show all main patterns as well as their relationships. This map as explained in Chapter 4 could be an inspiring source for building many different applications.

The next important step of the work was to describe some patterns in more detail with the pattern template contained in reference [20]. As a few instances of patterns, one analysis pattern: AlternativeManagement, and two design patterns: AnyProject and AnyRisk have been chosen for detailed descriptions. The pattern templates for these

three patterns in appendix A of this dissertation. There are also available pattern templates developed for some of related concepts that have been referred to (see chapters two and three). They can be reused for RAAM applications as well.

Finally, the three patterns chosen for detailed descriptions have been implemented. A few demos inserted in appendix B show the actual quality of software applications build on stability based pattern language. Although the developed demos are small examples, their flexibility and scalability is clearly evident. Java as an executable language and OCL as a formalization language were used for the implementation part.

3. Future Work

After introducing the important patterns in the RAAM area, they should be described in a standard template as demonstrated in Appendix A. However, preparing more pattern templates for all patterns, introduced in this PL is the most important work to be done to complete this work.

Other related patterns will be identified during the pattern descriptions as the second level designs. Defining such patterns and logging them in complete pattern templates is another important task to be done in the future.

Implementation of the patterns is another crucial task for the future. Chapter 5 is dedicated to preceding steps that fill the gap between the PL design and its implementation. The mentioned actions described in this chapter should be done accurately to ease the job of developers. On the other hand, coding some examples would be a useful hint for later reuses, which can be done both in programming languages and formalization ones. Appendix B includes some examples developed by Java as a programming languages and OCL as a formalization one.

References

- [1] Chao, H.P., (1995). Managing the risk of global climate catastrophe: an uncertainty analysis. *Risk Analysis*, 15, 69-78.
- [2] Fayad, M.E. and Altman A., (September 2001), Introduction to Software Stability, *Communications of the ACM*, 44(9), 95-98.
- [3] Fayad, M.E. (January 2002), Accomplishing Software Stability, *Communications of the ACM*, 45(1).
- [4] Fayad, M.E. (March 2002), How to Deal with Software Stability, *Communications of the ACM*, 45(3).
- [5] Fayad, M.E and S. Wu. (September 2002), Merging Multiple Conventional Models in One Stable Model, *Communications of the ACM*, 45(9).
- [6] Fayad, M.E. (2004), *Stable Analysis Pattern*, New York: Wiley & Sons, In progress.
- [7] Hattis, D. & Burmaster, D.E., (1994), Assessment of variability and uncertainty distributions for practical risk analysis, *Risk Analysis*, 14, 713-730.
- [8] Hamza H. and Fayad M.E., (September 2002), A Pattern Language for Building Stable Analysis Patterns, *9th Conference on Pattern Language of Programs (PLoP 02)*, Illinois, USA.
- [9] Hamza, H. and M.E Fayad. (September 2003), Towards a Patterns Language for Stable Software Patterns, *Proceedings of Pattern Language of Programs' 2003 (PLOP'03)*, Monticello-Illinois, USA.
- [10] Helton, J.C., (1994), Treatment of uncertainty in performance assessment for complete systems, *Risk Analysis*, 14, 483-511.
- [11] Howard, R.A., (1988), Uncertainty about probability: a decision analysis perspective. *Risk Analysis*, 8, 91-98.
- [12] Pate-Cornell M.E., (1996), Uncertainties in risk analysis: Six levels of treatment, *Reliability Engineering & System Safety*, 54(2-3), 95-111.
- [13] Chernoff, H. & Moses, L.E., (1959), *Elementary Decision Theory*, New York: Wiley.
- [14] Hacking, J., (1975), *The Emergence of Probability*, *Cambridge University Press*, Cambridge, UK.
- [15] Hoffman, F.O. & Hammonds, J.S., (1994), Propagation of uncertainty in risk assessments: the need to distinguish between uncertainty due to lack of knowledge and uncertainty due to variability. *Risk Analysis*, 14, 707-712.
- [16] EUROPA, *European agricultural management*, Retrieved June 20, 2004, from europa.eu.int/comm/agriculture/publi/insurance/index_en.htm
- [17] NASA, *Continuous Risk Management*, Retrieved June 22, 2004, from crm.nasa.gov/knowledge/tools.html
- [18] Item Software Inc. *Quantitative risk assessment system*, Retrieved June 12, 2004, from www.itemsoft.com/qras.html
- [19] Joint Technical Committee (Australia/NZ), (1995), Risk Management AS/NZS 4360, *Standards Australia/Standards New Zealand*.

- [20] M.E. Fayad, (December 2004), *Stable Design Patterns: True Solution Space*, New York: Wiley & Sons, In progress.
- [21] Koller Gelenn, (2000), *Risk Modeling For Determining Value And Decision Making*, Boca Raton, London, New York, Washington D.C.: Chapman & Hall/CRC.
- [22] Hertz David B. and Thomas Howard, (1983), *Risk Analysis And Its Applications*, New York: John Wiley & sons.
- [23] S. Focardi and C. Jonas, (1998), *Risk Management: Framework, Methods and Practice*, New Hope, Pennsylvania: Frank J. Fabozzi Associates.
- [24] B. S.Blanchard, W.J. Fabrycky, (1998), *Systems Engineering and Analysis*, Prentice Hall Inc. London, Toronto, etc.
- [25] Hattis, D. & Burmaster, D.E., (1994), Assessment of variability and uncertainty distributions for practical risk analysis, *Risk Analysis*, 14, 713-730.
- [26] Ripon Bhattacharjee, (May 2004), *Patterns Language For Real-Time Embedded Systems*, Master project dissertation.
- [27] Mohamed E. Fayad, (October 2003), Jayashree Rajagopalan, Anita Ranganath, AnyLog Stable Design Pattern, *The 2003 IEEE International Conference on Information Reuse and Integration (IEEE IRI' 03)*, Las Vegas, NV.
- [28] S. Shlien, (1990), Multiple binary tree classifiers, *Pattern Recognition*, 23, 757-763
- [29] R. E. Bryant, (Sep. 1992), Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams, *ACM Computing Surveys*, 24(3).
- [30] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, (October 1994), *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley.
- [31] D. Schmidt, M.E. Fayad, and R. Johnson, (October 1996) *Software Patterns, Communications of the ACM*, 39(10), 36-39.
- [32] M.E. Fayad, D. Schmidt, and R. Johnson. (September 1999), *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, New York: John Wiley & Sons.
- [33] M.E. Fayad, D. Schmidt, and R. Johnson, (September 1999), *Implementing Application Frameworks: Object-Oriented Frameworks at Work*, New York: John Wiley & Sons.
- [34] M.E. Fayad and R. Johnson. (October 1999), *Domain-Specific Application Frameworks: Experience by Industry*, New York: John Wiley & Sons.
- [35] Garry Froehlich, H. James Hoover, Ling Liu, and Paul Sorenson. , (September 1999), *Reusing Application Frameworks Through Hooks*, Chapter in *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. M.E. Fayad, et al (Editors), New York: John Wiley & Sons.
- [36] Alexander, C., Ishikawa, S., Silverston, M., Jacobson, M., Fiksdahl-King, I., and Angel, S., (1997), *A Pattern Language*. *Oxford University Press*, New York.
- [37] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal, , (1996), *Pattern-Oriented Software Architecture: A System of Patterns*, Wiley and Sons.

[38] Edited by James O. Coplien and Douglas C. Schmidt, (1995), *Pattern Languages of Program Design*, Addison-Wesley.

[39] Dirk Riehle and Heinz Zullighoven, (1996), Understanding and Using Patterns in Software Development, *Software Engineering*, Volume 1: The Development Process, 2nd Ed. 225-236.

Appendix A

Pattern Catalog

1- AlternativeManagement Analysis Pattern

AlternativeManagement pattern is provided to capture the core concept of the alternative management process and propose a stable analysis pattern to show the main concepts related to this pattern as well as the relationships among them. AlternativeManagement Analysis pattern structure would be scalable to accept area-dependent Industrial Objects so that it can be reused in any application area. AlternativeManagement Analysis pattern is created based on Software Stability Method (SSM) [4]. SSM is a new development and modeling method that commits to understanding the problem conceptually and identifying the main concepts to catch its core knowledge. The final pattern provided in this template is shown to be scalable and reusable in different domains with few efforts.

Introduction

Remember the last time when you wanted to buy a shirt. There were several items with different colors, styles, materials, and prices. Which one did you decide to buy? Why? A friend of mine selects her clothes based on a two-column list of pros and cons for each option. In fact she is famous for effective shopping because she considers all possibilities as well as the important decision factors before actual buying.

Alternative management is the process of choosing the best option when there are several alternatives to fit a need. It is a systematic way for the multi-aspect evaluation of all available options before making any decision. Alternative management starts with the identification of alternatives. Usually in this phase participants do not think about the value of their candidates but try to gather every possible one. The goal is to ensure that no alternative, even if it seems unlikely to be the best one at first glance, has been lost. Sometimes alternatives are developed automatically by some software applications. Then parties identify their definition of the ideal choice. This would be defined in terms of a series of evaluation factors such as price, beauty, and size, in the shirt example. Some of these demands must be met for a candidate to be eligible for selection, while the others are privileges that make an acceptable choice better. In our example, a shirt must be in the right size and within the affordable price range to be acceptable, but the beauty is something that you can describe as “the more, the better.”

After defining alternatives and evaluation factors, it is time to evaluate each alternative based on the defined factors. First we need to get rid of the alternatives that do not satisfy the minimum requirements for an acceptable choice. Then we need to measure each alternative and assign it a representative number per evaluation factor. These numbers show each alternative’s degree of desirability when it comes to a particular evaluation factor. However the factors are different in nature, so we would have a few incomparable values at this stage that should be combined into a unique measure of

desirability for each alternative. There are conceptually different ways to do so. One popular way is to assign an importance coefficient to each factor that shows the percentage of desirability that is caused by each unit of this particular evaluation factor. For example, if the importance coefficient of price per dollar is %10, the shirt, which is cheaper by one dollar, will be assumed %10 more desirable than. Another way to evaluate alternatives is to give priorities to the evaluation factors; in this case the alternative with the best value for the top factor is assumed the most desirable one. In case of ties, we would consider the next top evaluation factor and so on. The right evaluation mechanism for a particular project depends on its characteristics and goals.

One crucial aspect of the alternatives, to be studied carefully in the alternative management, is their risk value. Because of the considerable costs imposed by the risks, almost in any kind of project, we prefer alternatives with less risk values even if it means less desirability based on other evaluation factors. In fact, reliability, which is achieved by reducing the risks, can be counted as one of the most important evaluation factors in most projects. However, it is worth emphasizing the risk as a crucial base for alternative evaluation separately because of its importance. The risk value can be treated like other measure values in any evaluation mechanism.

This pattern is supposed to be general enough to cover all different areas such as art, literature, pure science, engineering, humanities and so on. It is also meant to work with any kind of alternatives as well as evaluation factors and mechanisms. As another consideration for AlternativeManagement, one can think of its participants, who are different in number, nature, and roles. So the pattern should support all these diversities and be flexible enough to model any kind of alternative management.

Pattern's Name: AlternativeManagement Analysis Pattern

Choosing between the available choices is a very common activity in every day life. We come across different alternatives in various positions in life and need to choose the best one based on some criteria in our minds. Although people have different mechanisms to make the final decision, they understand the meaning of alternative management and its common features and data items, which is in fact the core concept of *AlternativeManagement* Stable Analysis Pattern.

AlternativeManagement is a Stable Analysis Pattern or as we call them in Software Stability Model (SSM), an Enduring Business Theme (EBT). EBTs represent important concepts within the problem spaces that show system goals and missions. As a distinctive attribute, each EBT has a stable, enduring concept that does not change over the time or across different domains. They help designers understand the problem deeply and offer a stable problem analysis, which is the reason why we call this pattern, *AlternativeManagement* Stable Analysis Pattern.

Also Known As: None

Problem

AlternativeManagement is the process of selecting the best choice among available candidates that can substitute with almost similar performance. It deals with the measurement of existing alternatives based on the asset of evaluation factors and comparing them by an evaluation mechanism. *AlternativeManagement* stable analysis pattern is supposed to capture the core concept of alternative management in general to cover all common structures, functionalities, and relationships that may be reused in different applications with the same goal. This is not, however; an easy task besides the identification of basic assets of the area that should be addressed in any solution for this problem: alternatives, evaluation factors, and the evaluation method, issues like how to evaluate each alternative and how to compare different alternatives.

Context

Alternative solutions for the same problem exist in any field of knowledge from engineering to art and from complicated projects to everyday life. We all face some “which one is the best?” problems daily that may be systematically solved by step-by-step alternative management. Of course, this does not mean that the process is exactly the same; there are also differences in the alternatives we have as well as the bases and the way we evaluate each option. However common attributes and functionalities in a given alternative management process will be captured in this Stable Analysis Pattern (SAP).

Forces

AlternativeManagement pattern should resolve the following forces:

- Alternatives are different naturally. They may be components of a complicated science project or different ways to do a common chore. These alternatives may be identified in a brain storming session, derived from a history log from similar projects, or even generated by software applications. However these differences do not prevent us to seek commonalities in different instances of alternatives that we should carefully consider in the pattern.
- Alternatives are evaluated based on a series of factors that could be very diverse. They may be quantitative or qualitative and may be measured subjectively or objectively. Nevertheless they are our way of defining the ideal candidate in the situation. Beauty, performance, cost, and understandability are some common factors in engineering project alternative management. Any solution for the alternative management problem should consider all these differences carefully and design flexible ways to address any kind of evaluation factors properly.
- After alternatives are measured based on the quality factors we will have a set of values for each pair of alternative and evaluation factors. It is possible and very usual that no one beats all the others in all evaluation factors, so we need to combine all the values and come up with a unique desirability degree per alternative to compare them. Any formula that weights the evaluation factors and merges them together would be a possible evaluation method and the pattern is expected to consider all possibilities for the evaluation mechanism too.

- One of the most important instances of an evaluation factor is the reliability of the system, which has a direct relationship with the concept of the risk. It is a crucial task to measure the risk value of each alternative and enter it as an important decision factor in the final equation. This is the reason why one may emphasize more on the risk value of the alternatives in this pattern.
- Alternative management involves many different participants such as managers, engineers, end users, and customers who identify alternatives, evaluation factors, and the evaluation mechanism. These participants may be individuals and/or organizations and may work together or separately. The pattern has to cover all possibilities and keeps the applicability in all different cases.

Solution And Pattern Structure

Figure 20 shows the stable class diagram of the AlternativeManagement pattern:

Figure 20- Stable analysis pattern for AlternativeManagement

Of course, every pattern involved in this SDP has its own attributes and functionalities. To cover this extra information, a more detailed pattern structure was developed in Figure21.

Figure 21 - Detailed pattern structure for AlternativeManagement stable analysis pattern

Classes:

- *AnyAlternative*: Represents existing alternatives to be evaluated. It covers the wide varieties of such alternatives in type and nature.
- *AnyEvaluationMechanism*: Shows the process of evaluating an alternative, which includes getting rid of unacceptable options, measuring each alternative using the evaluation factors, and calculation of the final formula that combines these values into a unique measure of desirability.

Patterns:

- *AlternativeManagement*: Represents the process of alternative management. It governs all activities in the process and does the required checks.
- *AnyEvaluationFactor*: Represents any base for the evaluation of each alternative. These bases may be qualitative or quantitative but should be measured by a numerical metric.
- *AnyParty*: Represents any participant of the project that will work on an aspect. Managers, engineers, users, customers, and reviewers are instances of project participants. Reference [4] includes a stable design pattern for AnyParty.
- *AnyRisk*: Represents probable events that may influence each alternative and cause a negative impact on the project cost or performance.

CRC Cards

CRC cards define responsibilities, and collaborations for each class. They also name a role for each class, which is useful for identifying the class responsibility. Each class should have only one unique responsibility. The collaboration part shows class clients and services. Here are the CRC cards for this model's classes:

AnyAlternative (alternative)		
Responsibility	Collaboration	
Shows the existing options that can be substitute each other without major performance differences.	Clients	Services
	AnyRisk<<BO>> AnyEvaluationMechanism<<BO>> AlternativeManagement<<EBT>>	define()

AnyEvaluationFactor (evaluation factor)		
Responsibility	Collaboration	
Represents a base for evaluating the alternatives.	Clients	Services
	AnyEvaluationMechanism<<BO>> AnyParty<<BO>>	sift()

AnyEvaluationMechanism (evaluation mechanism)		
Responsibility	Collaboration	
Evaluates all alternatives based on the evaluation factors.	Clients	Services
	AnyEvaluationFactor<<BO>> AnyParty<<BO>> AlternativeManagement<<EBT>>	measure() evaluate()

AlternativeManagement (alternative management)		
Responsibility	Collaboration	
Selects the best possible choice among the available alternatives.	Clients	Services
	AnyEvaluationMechanism<<BO>> AnyParty<<BO>>	validate() select()

AnyRisk (risk)		
Responsibility	Collaboration	
Represents all risks that threaten the alternatives	Clients	Services
	AnyAlternative<<BO>>	identify() prioritize()

AnyParty (participant)		
Responsibility	Collaboration	
Participates in alternative management activities.	Clients	Services
	AnyEvaluationFactor<<BO>> AnyAlternative<<BO>>	control() participate() perform() request() initiate()

Consequences

AlternativeManagement Analysis pattern has the following benefits:

1. AlternativeManagement pattern covers a broad diversity of alternatives in type and nature. That's because it has a separate pattern for AnyAlternative that considers both commonalities and differences in the concept of the alternative. In fact the high level of applicability, scalability, and reusability of this pattern is partly because of its flexibility towards any kind of alternative that make it reusable in any context for any project.
2. This pattern includes a pattern for evaluation factors. That means it will be able to address all diverse factors that one may choose for evaluating the available alternatives. Considering the substantial number of candidates for evaluation factors and their issues with measurement, one can understand that having AnyEvaluationFactor as a separate BO would increase AlternativeManagement pattern's generality and understandability.
3. As you see in the last sections, each alternative is measured by all evaluation factors with endless possibilities for the combination of the resulting desirability values to calculate a unique comparable degree for each alternative. These include a variety from simply adding the scores together to using complicated formulas. However, AlternativeManagement pattern considers all options for the evaluation mechanism by adding a pattern that can be customized to the participants' need. It uses AnyEvaluationMechanism pattern to allow participants to identify and utilize the best method for evaluating alternatives.
4. Because any combination of participants is possible in the alternative management process, any pattern that makes concrete rules to restrict the role and number of stakeholders in a risk would certainly fail. Fortunately, AlternativeManagement pattern is very flexible when it comes to the participants. It uses AnyParty pattern to allow individual persons as well as big organizations to participate in the process at the same time with any possible roles.

The AlternativeManagement pattern has the following limitations:

- 1- It would be hard to capture all underlying issues related to different patterns that comprise *AlternativeManagement* pattern, at first glance. Other patterns were used in the final model that has its own problems and forces. These inside

concerns will be considered within the related pattern, which therefore, presents a second level of details for the original pattern. AnyParty and AnyRisk pattern-BOs are instances of the above-mentioned patterns for which we can provide separate pattern templates or refer to reusable stable patterns. However, one may consider this pattern incomplete because of such hidden issues.

- 2- At least for many engineers, understanding a conceptual model is not an easy task. In the presented pattern, there is no IO described in the pattern, which makes the pattern's understandability an issue for beginners. Attaching such IOs, which are implementation details for application-dependent classes, will narrow the applicability and makes the pattern usage obvious. Instead we preferred to keep the pattern's generality and tried to increase the understandability with carefully chosen names and tangible examples. The next sections are devoted to some applicability examples to clarify the usage of AlternativeManagement pattern in real applications.

Applicability

To illustrate the use of this pattern in different applications, two examples are presented in this paper. For simplicity purposes, and since we are interested in those parts of the application models that utilize this pattern, we concentrated only on the relevant aspects in each application.

Example1: The best path in the map

Sometimes there are several paths to reach a certain target starting from an initial point. Our goal usually is to find the closest path with less traffic. These alternative directions have different characteristics and we usually face some tradeoffs to select the best way. However some risks threaten our travel such as the risk of getting lost, having a car accident, and running out of gas. Usually the driver and passengers decide about the optimum path based on the priorities in particular situations. Figure 22 shows a design diagram for this problem using the AlternativeManagement Analysis pattern:

Figure 22- Stable design diagram for path-map problem

A substantial number of the people suffer from obesity. It decreases the self-esteem and agility in most persons and lowers their life quality in most cases. In order to resolve this issue, nutritionists offer a number of solutions with differences in their opinions. Each solution has its cons and pros and everybody should select the best one for him/herself based on his/her lifestyle, health situation, and income. Finding an easy, effective, affordable solution that has the minimum risk in terms of side effects is an important problem that can be solved by the use of the design diagram proposed in Figure 4. The following diagram is based on AlternativeManagement Analysis Pattern.

Conclusions

The software stability modeling creates a base for building stable and reusable software patterns. The combination of EBTs and BOs gives a stable pattern that can be reused to model the same problem in different applications. In addition, each EBT and BO in the SSM can be further modeled using stability concepts, and those models would be used to describe the second level design for software systems. The resulting patterns are stable and demonstrate efficient reusability while modeling different applications.

As shown in two examples, the use of AlternativeManagement pattern is not through analogy. The pattern can be spotted easily in any system dealing with the same concept by adding proper Industrial Objects to the developed pattern. Moreover, because each class in AlternativeManagement pattern has its clear role, the classes can also be used to form other patterns in appropriate context.

References

- [1] H. Hamza, A Foundation For Building Stable Analysis Patterns, Master thesis, University of Nebraska-Lincoln, 2002
- [2] M.E. Fayad. How to Deal with Software Stability, *Communications of the ACM*, 45(4), April 2002
- [3] M.E. Fayad, and A. Altman. Introduction to Software Stability, *Communications of the ACM*, 44(9), September 2001.
- [4] M.E. Fayad, H. Hamza, and M. Cline. Stable Software Patterns: Analysis, Design, and Applications. (Book in progress)
- [5] H. Hamza and M.E. Fayad. A Pattern Language for Building Stable Analysis Patterns, 9th Conference on Pattern Language of Programs (PLoP 02), Illinois, USA, September 2002
- [6] M.E Fayad. Accomplishing Software Stability, *Communications of the ACM*, 45(1), January 2002

2- AnyProject Design Pattern

AnyProject pattern is provided to capture the core concept of any project and propose a stable design pattern to show the main concepts related to this pattern as well as the relationships among them. AnyProject design pattern would be scalable to accept area-dependent Industrial Objects, so that it can be reused in any application area. AnyProject design pattern is created based on Software Stability Method (SSM) [4]. SSM is a new development and modeling method that commits to understanding the problem conceptually and identifying the main concepts to catch its core knowledge. The final pattern provided in this template is shown to be scalable and reusable in different domains with few efforts.

Introduction

Every day we wake up with a list of tasks to do. Some are as simple as a flat task, while others are more complicated using several resources, following more than one mission, or including a number of sub-tasks to be done. We usually refer to the latter group as projects regarding the area to which they belong. However, there are tremendous projects that are defined in national or international levels. These big projects usually demand intricate planning to arrange all aspects precisely, such as the Manhattan Project, the Fifth Generation Project, or even publishing a paper.

The Manhattan Project is a famous example. It was an effort during World War II in the United States to develop the first atomic bomb. This top-secret project lasted from 1939 until 1945 and utilized various resources under the direction of General Leslie R. Groves, Deputy Chief of Construction to the U.S. Army Corps of Engineers. It resulted in the successful production of nuclear bombs used in WW II.

The Fifth Generation Project began in 1981 as an attempt by the Japanese government to develop a new, more advanced generation of computers supposed to perform huge calculations utilizing massive parallelism. The project ended in 1992. The Fifth Generation computer system was aimed at becoming a new, low cost, often simpler technology, to displace the existing technology but ended up as a complete failure. The computers, operating systems and programs produced by the project only have academic interest these days.

According to the dictionary definition, "The word *project* comes from the Latin word *projectum* from *projicere*, "to throw something forwards" which in turn comes from *pro-*, which denotes something that precedes the action of the next part of the word in time and *jacere*, "to throw." The word "project" thus originally meant "something that comes before anything else is done." When the word was initially adopted, it referred to a plan of something, not to the act of actually carrying this plan out. Something performed in accordance with a project was called an object. This use of "project" changed in the 1950s when several techniques for project management were introduced: with this advent the word slightly changed meaning to cover both projects and objects. Now the term means: "A discrete collection of tasks, which are required to fulfill a specific set of

related goals, and/or resolve a specific development problem” as it comes in the same reference.

In today’s world project is a widely ranged concept that would cover almost any finite set of activities done to achieve one or more goals. The term is used from big research programs in space exploration to small art-craft activities. We even use the expression when we talk about spiritual programs like meditations. Although any project has its own characteristics that make it different from others, one can think of AnyProject as a common concept with clear definition and boundaries.

This pattern is supposed to be general enough to cover all different areas such as art, literature, pure science, engineering, humanities and so on.... It is also meant to work with any kind of deliverables and resources. As another consideration for AnyProject, one can think of project participants who are different in number, nature, and roles. So the pattern should support all these diversities and be flexible enough to model any kind of project.

Pattern’s Name: AnyProject Design Pattern

The term project is a familiar yet meaningful word that everybody knows what it refers to. Engineers as well as others have a lot of different experiences of its usage so they can understand the variety of instances perfectly. This can lead them easily to the commonalities among these instances, which is in fact the core concept of *AnyProject* design pattern.

AnyProject is a Design Pattern or as we call them in Software Stability Model (SSM), a Business Object (BO). BOs represent semi-tangible concepts within problem spaces that cooperate to fulfill a particular goal or mission. As a distinctive attribute, each BO has many different instances in real world with diversities in several aspects and some common features that are captured in the related BO. In order to remind this characteristic of BOs we added the word “Any” to the beginning of BO names, therefore we call this pattern *AnyProject Design Pattern*.

Also Known As:

The pattern may also be known as venture, endeavor, undertaking, enterprise, and job. However, there are minor differences among these terminologies. For this reason we prefer to use the expression AnyProject as the name of this pattern.

Problem

There are many Enduring Business Themes (EBTs) related to *AnyProject* as a general concept. However a project is always defined to satisfy one or several needs. It is a discrete collection of tasks, which are required to fulfill a specific set of related goals, and/or resolve a specific development problem. These tasks, goals, and problems may belong to a wide range of varieties but there are common structures, functionalities, and relationships that make us think about *AnyProject* as an independent asset and establish a desire to have a stable pattern that can explain the core knowledge of the main concepts

related to any project. More EBTs such as performance, effectiveness, and profitability may be related to AnyProject in different systems.

Context

We all start our days planning to do projects that hopefully resolve issues in our lives. These projects may involve various aspects of life such as work, study, and daily chores and they may be planned to achieve different goals. There are also differences in the way we model a project and specify its steps and rules. Complicated ways like using engineering tools and diagrams as well as simple ones like describing them verbally to your friend are examples of such modeling methods. However these projects are all about using some resources effectively to reach a milestone. All projects share the same central concepts that will be captured in this Stable Design Pattern (SDP).

Forces

AnyProject pattern should resolve the following forces:

- Projects are different themselves naturally. They include different tasks as the main steps. Going from home to school may involve walking through a specific path and taking rides at determined points. Writing an article, on the other hand, is a project including reading, writing, and reviewing tasks. However, these differences do not prevent us from seeking commonalities in different projects and these should be considered in the pattern so that it can address any project.
- Any project has a few resources to use. Time, money, personnel, equipments, etc are important examples of such resources. Since these resources are not infinitely available, project managers need to distribute them optimally to cover all requirements in the project. However these resources have a wide range of variety in their nature and limitations as well as in the allocation specifications in different projects. For instance, you budget your time to write an article far differently from your plan to spend your vacation money wisely in a trip. The pattern shall cover all kinds of resources along with their restrictions and allocation methods.
- Any project is defined to solve a problem in a research or development context. It sure is intended to satisfy a set of needs, which are the reasons why we start the project initially. Although these needs can be very different in various projects, they have a lot in common both conceptually and functionally. One can start a project to satisfy a personal or social need like repairing an old bicycle or improving morality in a society. The pattern is expected to consider all possible aspects of the initial needs too.
- Initial needs are general and qualitative concepts that show the main direction of the project. However, any project needs to have assets of measurable, binary milestones to define the path in more details. The milestones are the missions for project tasks and one can ensure the completeness of the project design by comparing them with the initial needs of the system. Take the example of

repairing a bicycle. The milestones in this case would be things like balancing the tire pressure, replacing the broken chain, and lubricating the break system.

- Any project is involved with many participants with wide differences. There are managers and staff involved in projects as well as users to utilize the results. There may also be qualifiers and reviewers to assure or check quality factors. The participants may be individuals and/or organizations and may work on the project separately or simultaneously. The pattern has to cover all possibilities and be applicable in all different cases.

Solution And Pattern Structure

Figure 24 shows the stable class diagram of the AnyProject pattern:

Figure 24- Stable design pattern for AnyProject

Of course, every pattern involved in this SDP has its own attributes and functionalities. To cover this extra information I have developed a more detailed pattern structure as it appears in Figure 25.

Figure 25- Detailed stable design pattern for AnyProject

Classes:

- *AnyProject*: Represents the project itself and contains all the behaviors and attributes of a general project.
- *AnyMilestone*: Shows missions of tasks in a project. Milestones should be detailed, measurable and binary. They are our tools to control the project's improvement.
- *AnyTask*: Represents each unique task determined by project modeling methods and together they form the project itself.

Patterns:

- *AnyNeed*: Shows the main reason(s) for the project initiation. They can be very different in type and nature.
- *AnyParty*: Represents any participant of the project that will work on one aspect of it. The managers, engineers, users, customers, and reviewers are instances of project participants. Reference [4] includes a stable design pattern for AnyParty.
- *AnyResource*: Covers any kind of resource on which a project will depend. It includes money, time, personnel, equipment and so on. For more details about this pattern see reference [7].

CRC Cards

CRC cards define responsibilities, and collaborations for each class. They also name a role for each class, which is useful for identifying the class responsibility. Each class

should have only one unique responsibility. The collaboration part shows class clients and services. Here are the CRC cards for this model's classes:

AnyMilestone (mission)		
Responsibility	Collaboration	
Shows the reasons a particular task is started.	Clients	Services
	AnyTask<<BO>>	verify() put-together() integrate() indicate()

AnyParty (participant)		
Responsibility	Collaboration	
Participates in any project.	Clients	Services
	AnyTask<<BO>> AnyResource <<BO>>	control() evaluate() participate() perform() analyze() request() deliver() initiate()

Need (need)		
Responsibility	Collaboration	
Ensures the participant that the project is exactly what they need and want	Clients	Services
	AnyProject<<BO>>	completenessCheck() performanceCheck() regulate() specify()

AnyProject (project)		
Responsibility	Collaboration	
Represents the project itself.	Clients	Services
	AnyResource <<BO>> Need<<EBT>> AnyTask<<BO>>	finish() feasibilityCheck() define()

AnyResource (resource)		
Responsibility	Collaboration	
Presents the available resources to be used in any project	Clients	Services
	AnyProject<<BO>> AnyParty <<BO>> AnyTask<<BO>>	schedule() staff() time() fund() required-tools() allocate()

AnyTask (project component)		
Responsibility	Collaboration	
Shows the main tasks to be performed in a project.	Clients	Services
	AnyProject<<BO>> AnyParty <<BO>> AnyMilestone<<BO>> AnyResource<<BO>>	state() done() require() identify()

Consequences

AnyProject design pattern has the following benefits:

1. AnyProject pattern covers many applications with very different natures. It can be used in any context for any kind of task set. Knowing the wide range of the usage of the project concept, one would really appreciate high level of applicability, scalability, and reusability for this pattern, which are direct results of its generality. AnyProject pattern holds the core concept of a project that is common among all different applications and domains.
2. Since there are countless possibilities for the combination of project participants, any pattern that makes concrete rules to restrict the role and number of stakeholders in a project would certainly fail. Fortunately AnyProject pattern is very flexible when it comes to project participants. It uses AnyParty pattern to allow individual persons as well as big organizations to participate in a project at the same time with any possible roles.
3. Using AnyResource class, this pattern considers all possible supplements that would be utilized in any project. Each resource has its own restrictions that should be considered in the scheduling process. Having a separate class for AnyResource will help users understand and organize resource issues better and customize the pattern for their purposes using devised hooks.

The AnyProject pattern has the following limitations:

1. It would be hard to capture all underlying issues related to different patterns comprising *AnyProject* pattern, at first glance. This is because we used other patterns in the final design model that have their own problems and forces. These inside concerns will be considered within the related pattern, which therefore, presents a second level of

details for the original pattern. AnyParty and AnyResource pattern-BOs are instances of the above-mentioned patterns for which we can provide separate pattern templates or refer to reusable stable patterns that are already developed as I did before for these two patterns in the section “Classes.” However one may consider this pattern incomplete because of such hidden issues.

2. At least for many engineers, understanding a conceptual model is not an easy task. In the presented pattern there is no IO described in the pattern, which makes the pattern's understandability an issue for beginners. Attaching such IOs, which are implementation details for application-dependent classes, will narrow applicability and makes the pattern usage obvious. Instead we preferred to keep the pattern's generality and tried to increase understandability with carefully chosen names and tangible examples. The next sections are devoted to some applicability examples to clarify the usage of AnyProject pattern in real applications.

Applicability

To illustrate the use of this pattern in different applications, two examples are presented in this paper. For simplicity purposes, and since we are interested in parts of application models that utilize this pattern, we concentrate only on the relevant aspects of each application.

Example1: The Manhattan Project

The Manhattan Project as I explained in the introduction was an effort to make an atomic bomb by the US Government and the US Army in early forties. After spending a considerable amount of time and money and by utilization of advanced laboratories and knowledgeable personnel, it finally finished successfully in 1945. Figure 26 shows that AnyProject SDP can completely model this project.

Figure 26- Stable design diagram for the Manhattan project

Example 2: Publishing A Paper

Publishing a paper is a project we often deal with, in our academic life. Like any other project we have a goal - to submit the paper - using particular resources and we have to work with several stakeholders to finish the project. As we captured the main concepts and established the relationships, we can easily scale the AnyProject SDP, introduced in this paper, to solve the problem with minimum effort.

Figure 27- Stable design diagram for publishing a paper

The above examples show that this model can be reused in any application in any domain with similar definitions and missions.

Usecase Description And Sequence Diagram

For more clarity we will explain usecase initiation of this application and provide the relative sequence diagram. Initiation is an important step of any project's development. It covers the project definition activities and resource allocations, as well as feasibility analysis. Here is a table including more details about this usecase.

Usecase Id.	UC 1.0		
Usecase Title	Project Initiation		
Actor	Role		
ProjectOwner ProjectManager	AnyParty AnyParty		
Class	Type	Corresponding Attributes	Corresponding Interfaces
AnyProject	BO	name, startDate	define()
AnyTask	BO	name, type, initiationDate, owner	feasibilitycheck(), identify()
AnyMilestone	BO	code, description	indicate()
AnyResource	BO	code, type, description, availabilityPeriod	allocate(), schedule()
AnyParty	BO	name, address, phone #, responsibility	initiate()
Need	EBT	title, description, idealist, importanceLevel	specify()
Usecase Description	<p>1- AnyParty<<BO>>, which can be ProjectManager<<IO>> or ProjectOwner<<IO>>, initiates the project by specifying a new Need<<EBT>> to be archived.</p> <p>2- Need<<EBT>> defines AnyProject<<BO>> at the first step.</p> <p>3- Then AnyProject<<BO>> identifies a set of tasks to be done in some order in the form of AnyTask<<BO>> patterns.</p> <p>4- each instance of AnyTask<<BO>> would indicate an instance of AnyMilestone<<BO>> to be its representative in work improvement management activities.</p> <p>5- AnyProject<<BO>> also allocates AnyResource<<BO>> required to do the job completely.</p> <p>6- AnyResource<<BO>> will schedule the available resources to cover all needs and send the result back to AnyProject<<BO>>.</p> <p>7- Based on the results of the former parts AnyProject<<BO>> will do the feasibility checks and send the result back to the Need<<EBT>>.</p> <p>Need<<EBT>> informs AnyParty<<BO>> about the result. It may try another idea later to propose another project that would satisfy the need.</p>		
Alternatives	-		

Table 4 – Usecase description template for project initiation

To illustrate this usecase graphically, we provide a sequence diagram, shown in the following figure:

Figure 28 - Sequence diagram for initiation usecase

Conclusions

The software stability concept forms a base for building stable and reusable software patterns. The combination of EBTs and BOs of one application can form a stable pattern that can be reused to model similar applications. In addition, each EBT and BO in the SSM can be further modeled using stability concepts, and those models can be used to form domain-independent patterns. The resultant patterns are stable and demonstrate efficient reusability while modeling different applications.

As shown in two examples, the use of AnyProject pattern is not through analogy. The pattern can be spotted easily in any system that includes the same concept. On the other hand, the developed pattern does not lose its generality, as we were able to apply it to model the same problem in two different applications. Moreover, one can realize that each class in AnyProject pattern has its clear role, which is independent from the application that this pattern will be used in.

References

- [1] H. Hamza, A Foundation For Building Stable Analysis Patterns, Master thesis, University of Nebraska-Lincoln, 2002
- [2] M.E. Fayad, How to Deal with Software Stability, *Communications of the ACM*, 45(4), April 2002

- [3] M.E. Fayad, and A. Altman, Introduction to Software Stability, *Communications of the ACM*, 44(9), September 2001.
- [4] M.E. Fayad, H. Hamza, and M. Cline. Stable Software Patterns: Analysis, Design, and Applications. (Book in progress)
- [5] H. Hamza and M.E. Fayad, A Pattern Language for Building Stable Analysis Patterns, 9th Conference on Pattern Language of Programs (PLoP 02), Illinois, USA, September 2002
- [6] M.E Fayad, Accomplishing Software Stability, *Communications of the ACM*, 45(1), January 2002
- [7] Ripon Bhattacharjee, Patterns Language For Real-Time Embedded Systems, Master project dissertation, May 2004

3- AnyRisk Design Pattern

AnyRisk pattern is provided to capture the core concept of any risk. A comprehensive solution for risk-related problems is shown as a pattern structure that catches the main relevant concepts along with the relationships among them. Adding proper Industrial Objects, which are domain dependent classes that customize a general pattern for a particular application, can easily scale AnyRisk design pattern. AnyRisk design pattern is created based on Software Stability Method (SSM) [4]. SSM is a new development and modeling method that intends to understand the problem conceptually and define the main concepts to catch its core knowledge. Two different examples from different application areas have been illustrated to show the excellent level of scalability and reusability of the proposed solution.

Introduction

We do a substantial number of activities everyday that need planning to achieve best results. On the other hand, there are some possible events that make these activities partial or total failures. We call these probable events with negative impacts on the projects risks and plan to understand them and reduce their threat to the system as much as possible.

One important characteristic of risks is that they are not certain events but the probability of their occurrence is considerable. In words we do not address events with very high or very low probability as risks in a project. As another attribute of the risks, one can mention that any risk has a negative impact on the system missions and properties. If a probable event does not affect the system in an important negative way we do not consider it as a risk. The most important risks are those who have substantial probability and severity. Figure 29 shows these two concepts visually:

Figure 29- Risk, probability of occurrence, and severity

Probability and severity of a risk are often combined into a mathematical variable, called risk value, which is an important base for computing risk priority for risk reduction activities.

Risks not only have a direct and certain negative effect on the project, which is considered under risk severity, but they also affect other risks of the project. This is because there are different types of dependencies among project activities so that failure of one would affect success chances of other project activities. One can think of project tasks, and therefore project risks, as a hierarchy structure of dependencies that can be logical, conventional, or competitive over common resources. It is wise for risk managers to give a higher priority to risks causing a major improvement in the overall risk value of the system and try to spend more resources to control them. So another key concept defined in risk theory is sensitivity of the project towards the risk. It simply shows the amount of improvement in the project's risk value when a particular risk probability decreases by some risk reduction activities.

As an example consider drought in a particular farmland. There is a specific probability for the occurrence of drought in a year, which is calculated by different methods and tools. If it is a drought year, the amount of agricultural products will be decreased and a negative impact would affect the profit of the farm. Since the probability and the severity of the event are substantial in central parts of the country, the drought is a crucial risk in these parts. Regarding severity of this risk, one should think of drought effects on other risks of the system. Drought, for example increases the risk of fire in the farm as well as the probability of diseases spreading. So the sensitivity level of the risk of drought would be high.

After recognizing and prioritizing the risks, managers will decide about actions to decrease the negative influence of the risks on the project, or as it is known in RAAM field: risk reduction activities. There are so many ways to reduce the risks and the proper way is chosen for each risk based on its priority and available resources. For less important risks we can accept the risk without any further action but continue monitoring it to be aware of any changes in its properties. For other risks we may replace related activities with other alternatives to avoid the risk or transfer it to another party. In most cases by the way, we change the situation in the project and hence the probability or severity of the risk to mitigate it.

In the drought example, the risk is important so we cannot accept it as it is. If we have any chance to buy land in other counties where the risk of drought is insignificant, we should do so. This way we can totally avoid the risk. Buying insurances is one of the most popular ways to transfer risks. For mitigating the risk of drought, farmers can buy artificial irrigation systems or plant more resistant species to reduce the risk value.

Pattern's Name: AnyRisk Design Pattern

The term risk is a familiar yet meaningful word that everybody knows what it refers to. Engineers as well as others have a lot of different experiences of its usage so they can understand the variety of instances perfectly. This can lead them easily to the commonalities among these instances, which is in fact the core concept of *AnyRisk* design pattern.

AnyRisk is a Stable Design Pattern or a Business Object (BO) as they are described in Software Stability Model (SSM) [1]. BOs represent semi-tangible concepts within

problem spaces that cooperate to fulfill a particular goal or mission. As a distinctive attribute, each BO has many different instances in real world with diversities in several aspects and some common features that are captured in the related BO. In order to remind this characteristic of BOs we add the word “Any” to the beginning of BO names. So we call this pattern *AnyRisk Design Pattern*.

Also Known As:

The pattern may be also known as danger, jeopardy, threat, hazard, menace, and peril. However, Risk is a well-known term in Risk Analysis, Assessment, and Management area with clear definition and boundaries so we prefer to use the expression AnyRisk as the name of this pattern.

Problem

AnyRisk usually is studied to reduce the cost that would be imposed on the project if unwanted events happen. This reduction maximizes project profitability, so it can be the most important EBT related to AnyRisk stable analysis pattern. AnyRisk itself is an unwanted event that is probable during the project’s lifetime and would cause damages to properties or missions of the project. Working with risks, engineers will be involved in a set of tasks, goals, and problems that may have a wide range of varieties. However there are common structures, functionalities, and relationships that make us think about *AnyRisk* as an independent asset and establish a desire to have a stable pattern that can explain the core knowledge of the main concepts related to any risk. AnyRisk is also related to other EBTs like Assessment, Measurement, and Allocation.

Context

We all start our days planning to do projects that hopefully will resolve issues in our lives. However each project is threatened by probable events that have negative impacts. These risks may relate to various aspects of the project. But they usually are the result of shortages in resources like money, personnel, and time or the lack of technical knowledge or experience. Anyway they always endanger either properties of the project or its targets and missions. In spite of all differences in various aspects, all instances of risk share the same central concepts, attributes and functionality that will be captured in this Stable Design Pattern.

Forces

AnyRisk pattern should resolve the following forces:

- Risks are different in nature, power, and domain. They may be categorized in categories like legal, technical, financial, and so on. They may have different level of fatality based on their value and may threaten the project in different ways. However these differences do not prevent us to seek commonalities in different risks and make a pattern to consider all common attributes and functionalities of the concept of risk.

- Any risk belongs to a project with some resources to be spent on a set of activities and a set of goals and targets. This is a loose definition for project and covers many activities in all different fields of art, technology, science, and even daily life. Any pattern for risk should consider the wide range of projects that may be threatened by risks. There should be enough flexibility in the pattern to cover all different ways of project definition and running so that the pattern can work with all instances of the project.
- Any risk is defined to maximize profitability of the system by planning for preemptive actions that would reduce the overall risk value of the system. These risk reduction activities vary in the nature, level of risk reduction, and demanded resources. However a good distribution of resources among possible risk reduction actions can maximize the project's benefit, which is the main target of the pattern.
- Dealing with any risk, we need to know about the fundamental characteristics of the risk before making any decision. These characteristics that illustrate the importance level of the risk, are the risk's probability, severity, and sensitivity. Definition of these three concepts has been completely discussed in the introduction section. However methodologies for determining and measuring them are so diverse that any pattern for risk should consider them as separate assets with enough flexibility to cover all diversities.
- Any risk involves many participants with a wide range of varieties. Managers and Engineers who work in the risk department, end users who utilize the project, and other parties who would review risk management activities to evaluate their effectiveness are examples of such participants. These participants may be individuals and/or organizations and may work on the risk separately or simultaneously. The pattern has to cover all possibilities and keeps the applicability in all different cases.

Solution And Pattern Structure

Figure 30 shows the stable class diagram of the AnyRisk pattern:

Figure 30- Stable design pattern for AnyRisk

Of course, every pattern involved in this SDP has its own attributes and functionalities. To cover this extra information I developed a more detailed pattern structure as it is shown in Figure 31.

Figure 31- Detailed stable design pattern for AnyProject

Classes:

- *AnyRisk*: Represents the risk itself and contains all behaviors and attributes of a general risk.
- *AnyProbability*: Stands for the probability of occurrence of the risk, which can be calculated in many different ways such as statistics, simulation, and expert estimation.
- *AnySeverity*: Includes all negative impacts of a particular risk if it happens in the project's lifetime. Usually all negative impacts of a risk are described in terms of monetary value so that they can be compared.
- *AnySensitivity*: Represents the effects of occurrence of a risk on other risks of the project. It shows the percentage of improvement in the overall risk value of the system, when a particular risk's probability decreases for a specific number of percentages.

Patterns:

- *AnyParty*: Represents any participant that will work on one aspect of the project. Managers, engineers, users, customers, and reviewers are instances of project participants. Project parties have an active role in design and execution of risk reduction plans. Reference [4] includes a stable design pattern for AnyParty.
- *AnyProject*: Project is here used as a wide concept. It covers any kind of activities that aim a particular target spending resources to reach it. AnyProject with this definition would be subject to different risks; as one can say risks are parts of a project. A pattern template has been developed in this chapter to cover all aspects of AnyProject stable design pattern.
- *RiskReduction*: Includes all actions to be taken to reduce probability or severity of a risk. Risk reduction activities usually increase system's profitability tremendously.

CRC Cards

CRC cards define responsibilities and collaborations for each class. They also name a role for each class, which is useful for identifying the class responsibility. Each class should have only one unique responsibility. The collaboration part shows class clients and services. Here are the CRC cards for this model's classes:

AnyRisk (risk)		
Responsibility	Collaboration	
Represents the risk itself	Clients	Services
	AnyProject<<BO>>	identify() prioritize() monitor()

AnySeverity (severity)		
Responsibility	Collaboration	
Presents the negative impact of the risk if it happens	Clients	Services
	AnyRisk<<BO>>	determine() measure()

AnyProject (project)		
Responsibility	Collaboration	
Represents the project that relates to the risk.	Clients	Services
	AnyParty <<BO>> AnyRisk<<BO>>	finish() feasibilityCheck() define()

AnySensitivity (sensitivity)		
Responsibility	Collaboration	
Evaluates the sensitivity of the whole project to the risk probability.	Clients	Services
	AnyRisk<<BO>> AnyProbability<<BO>>	compute()

AnyProbability (probability)		
Responsibility	Collaboration	
Presents the probability for the occurrence of the risk	Clients	Services
	AnyRisk<<BO>> AnySensitivity<<BO>>	calculate() modify()

RiskReduction (risk reduction)		
Responsibility	Collaboration	
Represent all actions taken to reduce the probability or the severity of the risk.	Clients	Services
	AnyRisk<<BO>> AnyParty <<BO>>	plan() avoid() accept() transfer() mitigate()

AnyParty (participant)		
Responsibility	Collaboration	
Participates in any project.	Clients	Services
	RiskReduction<<EBT>> AnyProject<<BO>>	control() evaluate() participate() perform() analyze() request() deliver() initiate()

Consequences

AnyRisk Stable Design pattern has the following benefits:

1. AnyRisk pattern covers the general aspects of various risks with different natures. It can be used in any context for any kind of project. Knowing the wide range of usage for the concept of risk, one would really appreciate the high level of applicability, scalability, and reusability of this pattern, which are direct results of its generality. AnyRisk pattern holds the core concept of a risk that is common among all different applications and domains.
2. Since there are too many possibilities for the combination of risk participants, any pattern that makes concrete rules to restrict the role and number of stakeholders in a risk would certainly fail. Fortunately AnyRisk pattern is very flexible when it comes to risk participants. It uses AnyParty pattern to allow individual persons as well as big organizations to participate in a risk at the same time with any possible roles.
3. Using AnyProbability, AnySeverity, and AnySensitivity classes as separate patterns, maintains the flexibility of this pattern and eases the future relationships of AnyRisk with other related patterns like assessment and measurement. Considering all possible methods and functionalities that would be utilized by these three patterns, separation of them would be a very positive step for data encapsulation and enhances the design quality as well.
4. Having a separate class for RiskReduction helps users understand and organize planning issues better and therefore customize the pattern for their purposes with the use of devised hooks by the classification of conceptually dissimilar issues in different patterns. It also provides a general model that can work with any kind of process for risk reduction variable from organization to organization and from field to field.

The AnyRisk pattern has the following limitations:

1. It would be hard to capture all the underlying issues related to different patterns that comprise *AnyRisk* pattern, at the first glance. This is because we used other patterns in the final design model that have their own problems and forces. These inside concerns will be considered within the related patterns, which therefore, present the second level of details for the original pattern. AnyParty and AnyProject pattern-BOs are instances of the above-mentioned patterns for which we can provide separate pattern templates or refer to reusable stable patterns that are already developed as I did before, for these two patterns in the “Classes” section. However one may consider this pattern incomplete because of such hidden issues.
2. At least for many engineers, understanding a conceptual model is not an easy task. In the presented pattern there is no IO described in the pattern, which makes the pattern's understandability an issue for the beginners. Attaching such IOs, which are implementation details for application-dependent classes, will narrow the applicability and makes the pattern usage obvious. Instead we preferred to keep the pattern's generality and tried to increase the understandability with carefully chosen names and

tangible examples. Next sections are devoted to some applicability examples to clarify the usage of Documentation pattern in real applications.

Applicability

To illustrate the use of this pattern in different applications, two examples are presented in this paper. For simplicity purposes, and since we are interested in those parts of application models that utilize this pattern, we concentrated only on the relevant aspects of each application.

Example1: Farming

There is substantial number of small farms with traditional agricultural methods, where a landlord employs a number of farmers to produce one or more agricultural products. However various types of risks like fire accidents, drought, and vermin threaten the farm. Usually no formal history is kept for risk occurrence data and farmers do not have access to professional sources for information. Instead, some experienced people estimate the risks based on their past practices. Figure 32 demonstrates the use of AnyRisk stable design pattern to simulate this system:

Figure 32- Stable design diagram for a farming project

Example 2: School Trip

Take a school trip as an example of projects. A teacher, school's principal, students and the bus driver go for a one-day trip for entertainment or educational reasons. The school experience shows that some risks such as getting lost, running out of gas, and injuries may be in different levels of importance. Fortunately, the past events have been logged in school reports carefully, so these reports can be used as an information resource helping to understand the value of each risk before making preemptive decisions. Figure 33 illustrates the reuse of AnyRisk stable design pattern to deal with the risks in a school trip:

Figure 33- Stable design diagram for a school trip

The above examples show that this model can be reused in any application in any domain with the same definition and missions.

Dynamic Description And Behavior Diagram

In order to show risk behavior during the system life cycle, I have prepared a State Transaction Diagram that illustrates different possible states of a risk and the transactions among them. Since this pattern is focused on risk reduction activities, the diagram only describes the states related to this EBT and the other states have been mentioned just to complete the cycle.

Figure 34- State transaction diagram for risk reduction activities

Conclusions

Software Stability Method directs engineers towards forming stable models that would serve as a base for building scalable, reusable, and understandable applications. The combination of the EBTs and BOs captures the core knowledge of the field that can be reused to model similar applications. In addition, each EBT and BO in the SSM can be modeled in more depth using stability concepts. The resultant patterns are stable and demonstrate efficient reusability in any area of knowledge.

AnyRisk pattern addresses many required attributes and functionalities that are conceptually related to the risk in details. On the other hand, the developed can be spotted easily in any system that includes the same concept, as we were able to apply it to model the same problem in two different applications. Moreover, each class in AnyRisk pattern has its clear role and unique responsibility that makes further implementations fast and easy.

References

- [1] H. Hamza, A Foundation For Building Stable Analysis Patterns, Master thesis. University of Nebraska-Lincoln, 2002
- [2] M.E. Fayad, How to Deal with Software Stability, *Communications of the ACM*, 45(4), April 2002
- [3] M.E. Fayad, and A. Altman. Introduction to Software Stability, *Communications of the ACM*, 44(9), September 2001.
- [4] M.E. Fayad, H. Hamza, and M. Cline. Stable Software Patterns: Analysis, Design, and Applications. (Book in progress)
- [5] H. Hamza and M.E. Fayad. A Pattern Language for Building Stable Analysis Patterns, 9th Conference on Pattern Language of Programs (PLoP 02), Illinois, USA, September 2002
- [6] M.E Fayad, Accomplishing Software Stability, *Communications of the ACM*, 45(1), January 2002

Appendix B

Implementation

Two simple applications have been implemented based on AnyProject and AnyRisk stable design patterns. They illustrate the reusability and scalability of the patterns in different areas of knowledge. A hard disk is attached to this document including the java code and required guidelines for running the applications.