# A Study of In-depth Issues Surrounding Object Oriented Languages using Object Oriented Design Patterns

Clarence J M Tauro
Christ University
Bangalore, India

N Ganesan
Director (MCA)
RICM, Bangalore

Krishna Priya R
Christ University
Bangalore, India

Bhavya F
Christ University
Bangalore, India

## ABSTRACT

In this competitive world, all enterprises depend on the IT as their active support for various purposes. As a result the software projects are becoming larger and more complex. Hence the developers face challenge of developing the complex software's more quickly. The best solution to this is the concept of Reuse [1]. Through object oriented analysis pattern it is possible to solve the problems occurring in software development through its reusable capacity [1]. Good OO designs are reusable and stable nature, this provides the ability of this pattern to be used in Other applications which share the same knowledge[7]. Patterns show you how to build systems with good OO design qualities. The main aim of "object oriented analysis pattern" is to provide expert solution to recurring business problem and to produce more reliable conceptual design. In this paper, we formally give an overview on the state of Object-oriented modeling using patterns and classification of analysis pattern and development on analysis pattern and also the applications of analysis patterns

## General Terms

Object Oriented Programming, Object Oriented Analysis Patterns.

## Keywords

Design pattern, reusable, analysis patterns, development, applications, Unified Modeling Language, Flexible Pattern Oriented Modeling, Validation and Optimization Method.

## 1. INTRODUCTION

The main idea of Object oriented Analysis is to study and model a given problem domain, user requirements by means of identifying and defining classes and their objects found in the specified domain. It mainly focuses on what the system should rather than how it should do. A pattern is a recurring motif, an event or structure that occurs over and over again[14]. Design patterns are a popular and successful means of implementing flexible and reusable software systems [11][12]. It describes the communication of objects and classes that are customised to solve a general design problem in a particular context[15]. Among these Pattern analysis is one of the forms of Object Oriented Analysis. It is a general reusable solution to solve the problems occurring in software design [1]. And also in software development, conceptual models are usually used to develop good understanding of the main concepts in the problem domain and also facilitates communication between developers and stake holders[5]. Patterns are basically a description or template for how to solve the problems occurring in many different situations. Object-oriented design patterns basically shows the relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved. Applying design pattern enables developers to reuse it to solve a specified designing issue. Design patterns help designers communicate architectural knowledge and also help people learn a new design paradigm, various patterns have been developed for each phase of software development that involves the concept of re-use: analysis patterns for the analysis phase, design patterns for the design phase. The distinguishing feature of patterns is that they provide expert knowledge for certain problems that have been formatted for re-use, allowing the faster development of more flexible applications which, through the use of tried and tested solutions. Effective use of this helps in enormous cost reduction since we avoid many possible faults which are very crucial in development. In this section we will overview of state of object oriented modeling and classifications of analysis patterns and development on analysis pattern and also the applications of analysis patterns.

## 2. OVERVIEW OF THE STATE OF THE OBJECT ORIENTED MODELLING

The basic idea behind OOA is to analyze a problem domain, and develop a conceptual model that can then be used to complete the task. While analyzing a particular problem the developer should consider what requirements needs to be met by the system. The conceptual model that results from OOA will typically consist of a set of use cases, interaction diagrams etc. As denoted by its name an OOAP regards itself as a pattern used in the OOA. It is basically an abstraction of a group of related generic objects with expected relationships which is likely to be helpful again and again in object oriented development [2]. Analysts can reuse and incorporate analysis patterns into their work to address their problems related to their application area. Hence they can effectively utilize their time and resources consumed in the conceptual modeling. Also, analysis patterns can allow other analysts to benefit from the experience of pattern designers

Benefits brought by OOAP are:

- Concept of Reuse – This is one of the major benefits. By reusing the existing designs the analysts can solve various repeated problems. They need not have to invent the solutions again. They will just need to reuse it according to their application needs. This can minimize the effort, as well as time and cost.
- Establish common terminology – from a solution readability and understandability perspective, analysis patterns provide a common base vocabulary and a common viewpoint of the problem for system analysts.
- Promote modeling quality – as promoting proven knowledge and solution to a recurring business problem, analysis patterns can give some indications to entire stakeholders about the quality of the overall conceptual model. Particularly, system analysts get the benefit of learning from the experience of others. A preliminary study has highlighted the effectiveness of applying analysis patterns in helping the analyst identity missing classes, associations and aggregations [2].

## 3. CLASSIFICATION OF ANALYSIS PATTERN

As there is increase in the number of patterns, it is very important to develop proper methodologies and techniques on how to classify them. Domain analysis makes a necessary contribution in supporting systematic reuse[10]. Pattern classification is the organization of patterns into groups of patterns sharing the same set of properties [1]. There are various criteria based on which the patterns are classified like discipline, domain, paradigm, scope, purpose etc. Through design patterns it is possible to solve the problems occurring in design issue. Design patterns help designers communicate the structural knowledge, help people learn new design categories and help new developers by committing errors and facing difficulties that have been traditionally learned only by costly experiences. There are various design patterns like Strategy Pattern, Sampling Pattern, Iterator Pattern, Adapter Pattern and Façade Pattern.

### 3.1 Sampling Pattern

This pattern represents the process of selecting a small portion or piece of items as a sample to represent a larger item or group of items [7]. Sampling is a widely used term that has several built-in essential characteristics, such as its capacity to cover multiple areas of application, its ability to enclosed distinct selection methods within its core mechanism for sampling, and the sampling capability itself which provides the base for today’s used sampling techniques

### 3.2 Strategy Pattern

Strategy Pattern is an encapsulation of algorithms. It basically separates algorithm itself and assigns different objects to them. As shown in Fig 1, generally Strategy Pattern encapsulates a family of algorithms into a Strategy of classes as the subclass of an abstract Strategy super class. The strategy pattern uses composition instead of inheritance and also in strategy pattern behaviors are defined as separate interfaces and specific classes that implements these interfaces. Specific classes encapsulate these interfaces. This allows better decoupling between the behavior and class that uses the behavior.

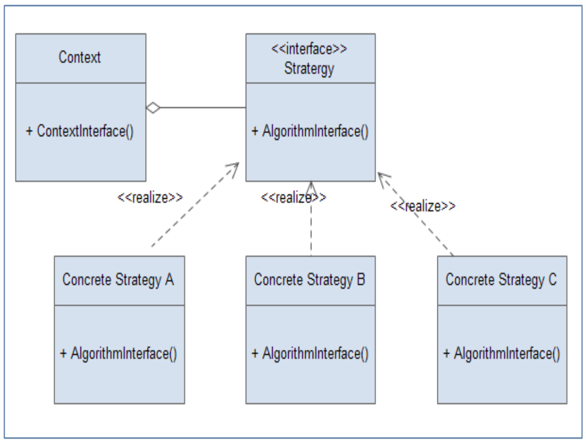

**Fig 1: Strategy Pattern Diagram**

The significant drawback of this approach is behaviors must be declared in each new sub class and managing of these behaviors increases greatly as the number of model increases and requires code to be duplicated across models and also it is not easy to determine exact nature of the behavior for each model.

### 3.3 Iterator Pattern

The Iterator pattern provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation and it also gives you a way to step through the elements of an aggregate without having to know how things are represented under the covers. The effect of using iterators in your design is very important: once you have a uniform way of accessing the elements of all your aggregate objects, you can write polymorphic code that works with any of these aggregates, without caring if the elements are held in an Array or ArrayList, as long as it can get hold of an Iterator. The other important impact on your design is that the Iterator Pattern takes the responsibility of traversing elements and gives that responsibility to the iterator object, not the aggregate object. Fig 2 shows the architecture of Iterator pattern. Here remove method is considered optional where it is not necessary to provide remove functionality but we need to provide the method because its part of the Iterator interfaces. The primary purpose of Iterator is to allow a user to process every element of a container while isolating the user from the internal structure of the container.

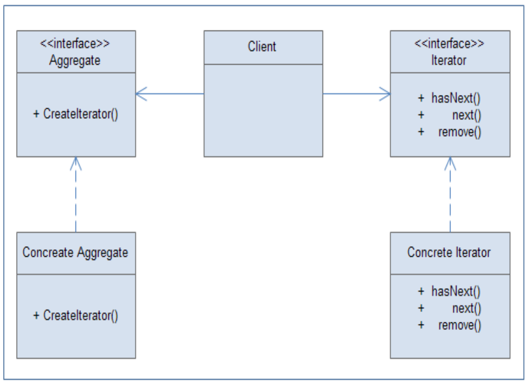

**Fig 2:Iterator Pattern class diagram**

## 3.4 Adapter Pattern

The Adapter Pattern converts the interface of a class into another interface the clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces. The job of implementing the adapter is really proportional to the size of the interface you need to support as your target interface.

There are two kinds of adapters: object adapters and class adapters [6]. The two diagrams specified in Fig 3 and Fig 4 look similar. The only difference is that with class adapter we subclass the Target and the Adapter, while with object adapter we use composition to pass requests to an Adapter. There are some issues to be mentioned while considering object adapter and class adapter pattern.

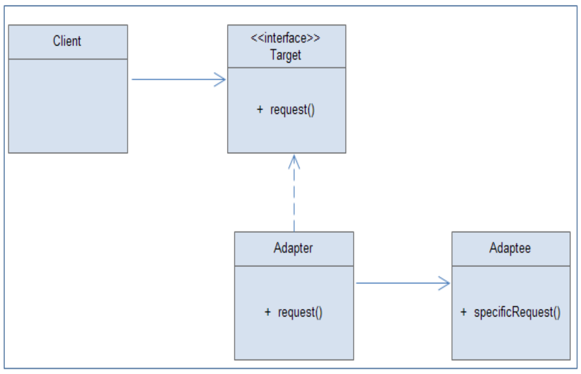

**Fig 3:Object Adapter Pattern Class Diagram**.

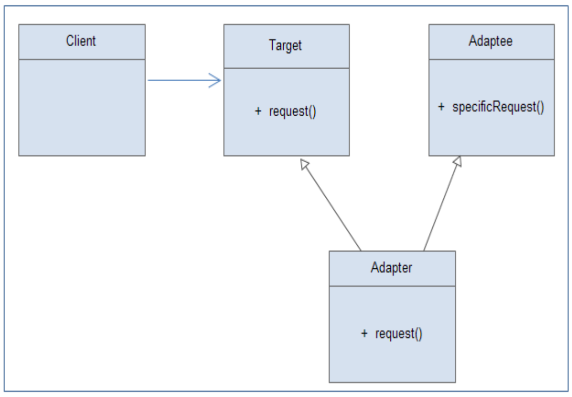

**Fig 4: Class Adapter Pattern Class Diagram**

### *3.4.1 Issues in Adapter Pattern*

Object adapter pattern use composition, it can not only adapt an adapter class, but any of its subclasses. Class adapter pattern do have trouble with that because it's committed to one specific adapter class, but it have a huge advantage because it doesn't have to re implement its entire adapter. Class adapter pattern can also override the behavior of my adapter if it need to because it's just sub classing. In object adapter pattern, we like to use composition over inheritance; class adapter may be saving a few lines of code, but all class adapter is doing is writing a little code to delegate to the adapter. Object adapter can keep things flexible. Using a class adapter there is just one of that, not an adapter and an adapter, which means efficiency. In general the adapter acts as the middleman by receiving requests from the client and converting them into requests that make sense on the vendor classes, without changing your existing code .The Adapter Pattern's role is to convert one interface into another. While most examples of the adapter pattern show an adapter wrapping one adapter, we both know the world is often a bit messier. So, you may well have situations where an adapter holds two or more adapters that are needed to implement the target interface.

## 3.5 Fascade Pattern

The Façade Pattern provides a unified interface to a set of interfaces in the sub-system. It provides a higher level interface that makes the subsystem easier to use. It also allows us to avoid tight coupling between clients and subsystems, and also helps us adhere to a new object oriented principle. Façades don't "encapsulate" the subsystem classes; they merely provide a simplified interface to their functionality. The subsystem classes still remain available for direct use by clients that need to use more specific interfaces. This is a nice property of the Façade Pattern: it provides a simplified interface while still exposing the full functionality of the system to those who may need it. As shown in Fig 5, façade is free to add its own "smarts" in addition to making use of the subsystem and also allows to decouple the client implementation from any one subsystem.

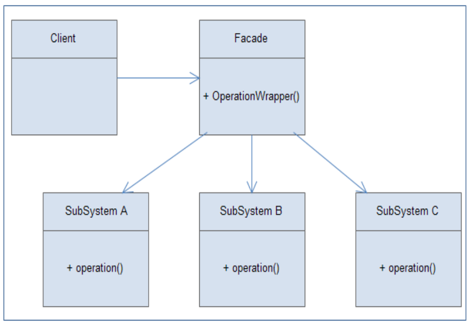

**Fig 5:Façade Pattern Class Diagram**

## 4. DEVELOPMENT OF ANALYSIS PATTERN

There are four main approaches that have been used to develop analysis patterns. They are the direct approach, specialization approach, analogy approach, and stability approach. The Fowler's (1996) analysis patterns belong to the direct approach category [2]. They are not generalized or abstracted further after they are identified. In the specialization approach, identified patterns are abstracted so that they are easier to apply themselves in similar and related applications than those patterns formed by direct approach. Analysis patterns described in Coad et al.'s belong to this approach. In the analogy approach, patterns are abstract to construct templates such as "Resource Rental" pattern (Braga et al., 1998) [3]. An analogy between the analysis pattern and the new application is conducted in order to adopt the identified pattern template to fit into the new application. Finally, the stability approach is a layered approach for developing new application. All these different approaches share the common theme of capturing core knowledge of the real-world problem for future modeling [2]. The Unified Modeling Language (UML), can be used to represent structural and behavioral information as part of commonly occurring object analysis patterns[8]. We can also specify design pattern transformations to evolve pattern instances like refactoring transformation and Design pattern transformation [13].

## 5. APPLICATION OF ANALYSIS PATTERN

There are various applications of analysis pattern. Code pattern is one of the analysis patterns which plays an important role in understanding the run time behaviors of OO systems[9]. Application of analysis pattern can be shown in certain methods. And these methods are presented in the simplified software development process which is shown in the Fig 6.

This consists of the analysis of the requirements, the creation of the analysis model, the creation of the design model, and the implementation. The methods presented develop the creation phase of the analysis model to a higher degree, with the analysis pattern application explicitly not being limited only to the static aspects of the model, but also extending to the dynamic ones.

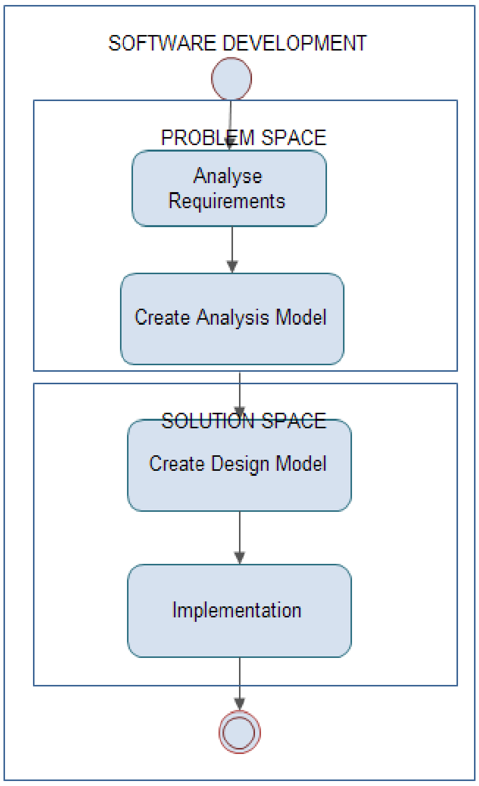

**Fig 6: Simplified Software development process**

Flexible Pattern Oriented Modeling Method (PMM), as well as the Validation and Optimization Method (VOM) are the two basic methods of the application of analysis patterns. All the methods that have been previously introduced can be subsumed under them. They were named according to their properties as they affect the way of pattern application. They are illustrated in the form of a UML activity diagram. Both methods require the availability of one or more sources of analysis patterns (e.g. pattern catalogue) they are, however, designed in such a way that they allow modeling even when there is no support from analysis patterns available.

### 5.1 Flexible Pattern-Oriented Modeling Method (PMM)

The Fig 7 shows the sequence of the Flexible pattern –oriented modeling method. This method is characterized by the integral use of analysis patterns in the analysis model development. It allows the use of analysis patterns at every stage of the creation of the model.

The method starts after the requirements analysis and, depending on the available patterns, begins either with the identification of the central classes of the model (identify central classes), or directly with the determination of a part of requirements for which a pattern is searched (select new part of requirements). The first alternative turns out to be useful for specific pattern catalogues. If the patterns of COAD et al. (1995) are used [3], their low complexity requires that the process begins with the modeling of the central classes, and the model is only completed afterwards with the support of patterns. If no matching pattern can be found for the requirements under consideration (no matching pattern found), the corresponding part of the model has to be modeled by hand (model selected part of requirements from scratch). If, on the other hand, a matching pattern is found (matching pattern found), then this is instantiated (instantiate analysis pattern) and subsequently integrated into the model (integrate analysis pattern into model). Then either the search for another analysis pattern begins, (analysis model not complete) or the model is complete and the design phase can be commenced. As we shall see later the PMM combines the known approaches for the application of analysis patterns from the literature in a flexible manner. Particular value was put on the method allowing it to be adapted to the available analysis patterns at every stage of the model development. In particular the PMM can (formally) also be applied if only a pattern catalogue of limited size is available.

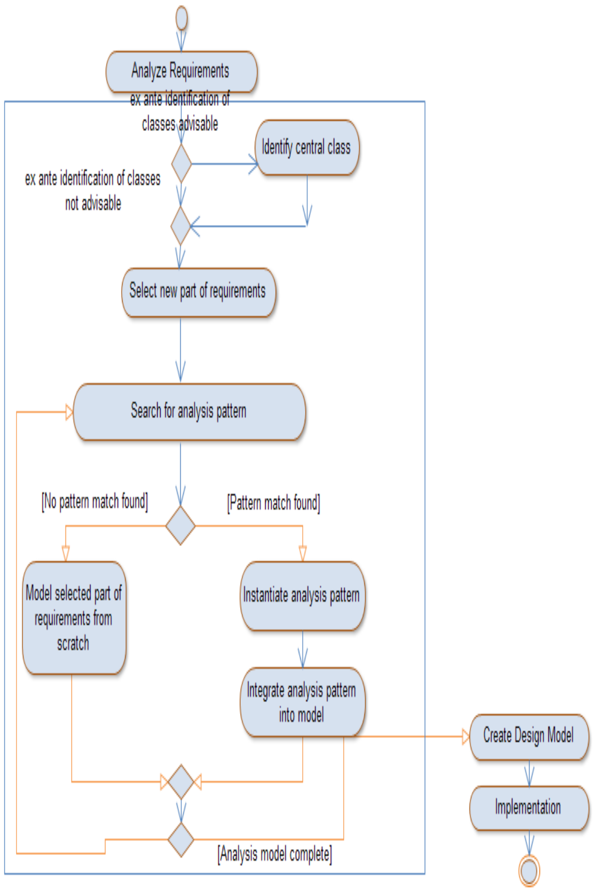
Analyze Requirements
ex ante identification of classes advisable
Identify central class
ex ante identification of classes not advisable
Select new part of requirements
Search for analysis pattern
[No pattern match found]
[Pattern match found]
Model selected part of requirements from scratch
Instantiate analysis pattern
Integrate analysis pattern into model
Create Design Model
Implementation
[Analysis model complete]

**Fig 7: Flexible Pattern oriented Modeling method**

## 5.2 Validation and Optimization Method (VOM)

The sequence of the Validation and Optimization Method (VOM) is shown in Fig 8. With this method analysis patterns are applied only after completion of the analysis model; the existing model is then validated and optimized with the help of matching analysis patterns. The creation of the analysis model and the review of the model with analysis patterns do not have to be carried out by the same people. The method can also be applied, therefore, in order to improve older models or models that were created by others. For example, when advisers join a project that has already been started, with VOM they can check the existing analysis model. After the requirements analysis the analysis model is modeled without pattern support (model selected part of requirements from scratch). This activity is continued until the model appears to be complete (analysis model complete). Finally, the model is validated and optimized with analysis patterns. All the patterns that match the problems solved in the analysis model have to at first be searched for – e.g. in a pattern catalogue (Search for analysis pattern). If no matching patterns are found (no matching patterns found), the design phase is commenced. If at least one matching pattern is found, then the first pattern (either the only one or the most suitable one) is compared to the model in order to detect deficits and find opportunities for optimization (compare model to one analysis pattern). If an enhancement is necessary or possible (enhancement necessary), it is carried out (enhance model). If no Enhancement is necessary; the pattern is marked in the model in order to improve the documentation and the comprehensibility (mark pattern in model). If the review of the model is complete, with all patterns found having been applied, the design phase can be commenced (model Reviewed completely). If the review is not yet complete, the model is compared to the next pattern, and so on. With this method analysis patterns are applied only after completion of the analysis model; the existing model is then validated and optimized with the help of matching analysis patterns. The creation of the analysis model and the review of the model with analysis patterns do not have to be carried out by the same people. For example, when advisers join a project that has already been started, with VOM they can check the existing analysis model.

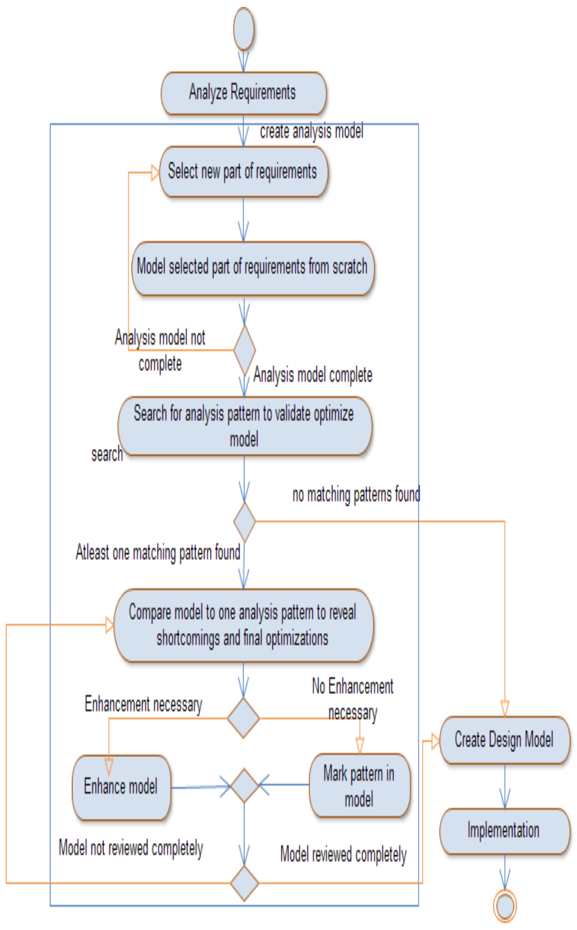
Analyze Requirements
create analysis model
Select new part of requirements
Model selected part of requirements from scratch
Analysis model not complete
Analysis model complete
Search for analysis pattern to validate optimize model
search
no matching patterns found
Atleast one matching pattern found
Compare model to one analysis pattern to reveal shortcomings and final optimizations
Enhancement necessary
No Enhancement necessary
Enhance model
Mark pattern in model
Create Design Model
Implementation
Model not reviewed completely
Model reviewed completely

**Fig 8: Validation and Optimization method**

## 5.3 Example for the Application of Analysis Patterns

### 5.3.1 *Party Relationship Pattern*

Party relationship pattern describes the parties and the relationships between parties in a trading community. Various problems that arise when we consider the context where organizations in a trading community need to interact with various other organizations. The Companies or organizations need to interact with many other organizations or individuals to conduct their business. Those organizations may have complex relationships with the organization and with each other. How do we model the complex relationship between parties so that the company knows the answers to the following key questions at all times: Who are my customers? How are they related to each other? What are their characteristics? Who are my competitors? Who are my partners? Who are my suppliers?

The solution for the above problem is affected by the following forces:

- We need to know how other parties are related to our organization so our interactions with them are appropriate and effective.
- Parties can be individuals or organizations, and we want to consider both types.
- An organization is itself a party.

- Parties can be related to each other in more than one way in a peer or hierarchical fashion.
- A party can have many relationships with the company, and furthermore, the relationships are dynamic, they can change at any given time.
- Relationships are reciprocal; they can be organization-to-organization, person-to-person, or organization-to-person.
- We need to model inter- and intra- organization relationships, and non-business relationships.
- Spouse Of or Child Of are examples of non-business relationships.
- We need to describe any type of relationship, including the ability to capture branches, competitors, resellers, business partners, etc.

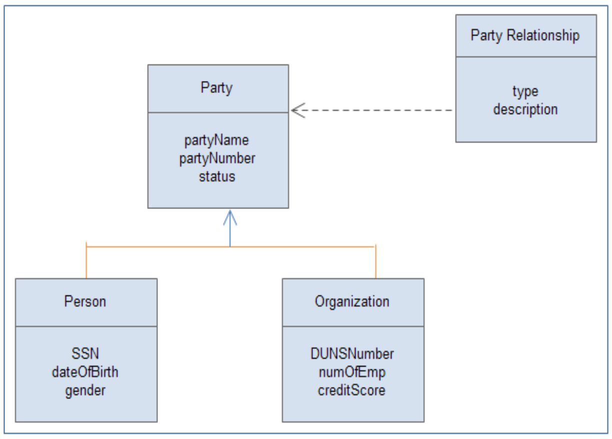

**Fig 9: Class diagram for the Party Relationship pattern.**

The Fig 9 describes the class diagram for the Party Relationship Pattern. Here Party can be a Person or an Organization that is of interest in a business context. Person is a unique individual of interest to the organization. Organization is a legal entity recognized by some government authority, i.e. a branch, a subsidiary, a legal entity, a holding company, etc. Party relationship links two Parties to indicate the nature of the relationship between them. This association may also indicate the direction of the relationship, superior or subordinate, as well as their roles in the relationship. For example, in an employee/employer relationship, employee is a role while and employer is another role. Some example relationships are: Client of/Contractor to, Supplier to/Distributor for, Seller to/Customer of, Reports to/Manager of, Employer of/Employee of, and Partner of.

## 6. CONCLUSION

In this paper we have given an overview on the importance of pattern analysis mainly in reusing the existing design patterns and finding the solutions for various repeatedly occurring problems in software design. We have explained about the different steps involved in analyzing a problem and developing a design pattern and also the classifications of patterns. We have also given a few commonly used analysis patterns and also described various methods used to solve the repeatedly occurring problems.

## 7. REFERENCES

[1] Jiang Shuai; Mu Huaxin; "Design patterns in object oriented analysis and design," Software Engineering and Service Science (ICSESS), 2011 IEEE 2nd International Conference on, vol., no., pp.326-329, 15-17 July 2011

[2] Felix Leung, Narasimha Bolloju; "Object-oriented Analysis using Patterns", http://www.pacis-net.org/file/2005/398.pdf.

[3] Nicolas Blaimer, Andreas Bortfeldt, Giselher Pankratz "Patterns in Object-OrientedAnalysis", http://www.fernunihagen.de/wirtschaftswissenschaft/download/beitraege/db451.pdf

[4]Mei Fullerton, Eduardo B. Fernandez, "Analysis Pattern for Customer Relationship Management (CRM)", http://www.cse.fau.edu/~security/public/docs/CRMPattMar05.pdf

[5] Chen, Y.; Hamza, H.S.; Fayad, M.E.; , "A framework for developing design models with analysis and design patterns," Information Reuse and Integration, Conf, 2005. IRI -2005 IEEE International Conference on. , vol., no., pp. 592- 596, 15-17 Aug. 2005 doi: 10.1109/IRI-05.2005.1506538

[6] Xiaoxi Chen; Jia Chen; Shengwen Zhang; Lili Sui; , "Application of adapter pattern in container ship stowage system," Industrial and Information Systems (IIS), 2010 2nd International Conference on , vol.1, no., pp.120-123, 10-11 July 2010 doi: 10.1109/INDUSIS.2010.5565897

[7] Sanchez, H.A.; Binbin Lai; Fayad, M.E.; , "The sampling analysis pattern," Information Reuse and Integration, 2003. IRI 2003. IEEE International Conference on , vol., no., pp. 601- 608, 27-29 Oct. 2003 doi: 10.1109/IRI.2003.1251472

[8] 12Konrad, S.; Cheng, B.H.C.; Campbell, L.A.; , "Object analysis patterns for embedded systems," Software Engineering, IEEE Transactions on , vol.30, no.12, pp. 970- 992, Dec. 2004 doi: 10.1109/TSE.2004.102

[9] Chung-Chien Hwang; Shih-Kun Huang; Deng-Jyi Chen; Chen, D.T.K.; , "Object-oriented program behavior analysis based on control patterns ," Quality Software, 2001. Proceedings.Second Asia-Pacific Conference on , vol., no., pp.81-87, 2001 doi: 10.1109/APAQS.2001.990005

[10] Cohen, S.; Northrop, L.M.; , "Object-oriented technology and domain analysis," Software Reuse, 1998. Proceedings. Fifth International Conference on , vol., no., pp.86-93, 2-5 Jun 1998 doi: 10.1109/ICSR.1998.685733

[11] Mens, T.; Tourwe, T.; , "A declarative evolution framework for object-oriented design patterns," Software Maintenance, 2001. Proceedings. IEEE International Conference on , vol., no., pp.570-579, 2001 doi: 10.1109/ICSM.2001.972774

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995

[13] C. Kramer and L. Prechelt. Design recovery by automated search for structural design patterns in object-oriented software. In Proc. Working Conf. Reverse Engineering, pages 208–215, 1996.

[14] Nierstrasz, O.; Demeyer, S.; , "Object-oriented reengineering patterns," Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on , vol., no., pp. 734- 735, 23-28 May 2004 doi: 10.1109/ICSE.2004.1317511

[15] Dori, D.; Perelman, V.; Shlezinger, G.; Reinhartz-Berger, I.; , "Pattern-based design recovery from object-oriented languages to object process methodology," Software - Science, Technology and Engineering, 2005. Proceedings. IEEE International Conference on , vol., no., pp. 77- 82, 22-23 Feb. 2005 doi: 10.1109/SWSTE.2005.1