International Journal of Engineering & Technology
CENTRE OF PROFESSIONAL RESEARCH PUBLICATIONS
ISSN 2049-3444
International Journal of Engineering and Technology Volume 1 No. 3, December, 2011

Comparative Study in Utilization of Creational Design Patterns in Solving Design Problems

R.M.Noorullah
Department of CSE, Amina Institute of Technology, Shamirpet, Hyderabad

ABSTRACT

Design Patterns solve specific design problems and make an object oriented design more flexible and reusable. They help designers reuse several designs, to choose design alternatives and to avoid alternatives that compromise reusability. To solve various design problems Creational, Structural and Behavioral patterns are used. In the present paper comparative study in utilization of Creational Design Patterns proposed to solve design problems by considering how they all encapsulate knowledge and hide instances of these classes. A critical study made for utilization of these patterns in solving design problems with some case studies and also relationship among these patterns presented with the help of design tools.

Key words: *Design Patterns, Abstract Factory, Singleton, Factory Method, Prototype, Builder*

1. INTRODUCTION

Design Patterns are descriptions of communicating objects and classes that are customized to solve a design problem at a particular context. It identifies the participating classes and interfaces, their roles, collaboration and distribution of responsibilities. Design Patterns solve specific design problems and make an object oriented design more flexible and reusable. They help designers reuse several designs, to choose design alternatives and to avoid alternatives that compromise reusability. To solve various design problems Creational, Structural and Behavioral patterns are used. Each Design pattern focuses on a particular OO Design problem or issue. Design patterns will be discussed by its name, classification, intent, structure, participants and by its implementations. They vary in their granularity and level of abstraction. Creational, Structural and Behavioral Patterns are various types of design patterns based on their purpose. In Section 2 Object Oriented Design Heuristics, in section3 Description and Selection of a Design Pattern to solve a design problem, in section 4 Description of common causes of design problem and proposed solution by using Creational design patterns, in section 5 step by step approach to use a right design pattern to solve design problem, in section 6 Study of Utilization of Creational Design patterns in solving design problems with real systems implementation, in section 7 relationships among Creational Design Patterns and Conclusion in section 8.

2. OBJECT ORIENTED DESIGN HEURISTICS

OOD Heuristics are used for design improvement. They are language independent and allow us to rate the integrity of a software design. The heuristics cover important topics ranging from classes and objects to physical OOD. Synergy will exist between design heuristics and design patterns. Heuristics can highlight a problem, while patterns can provide the solution. These heuristics will deal with classes and objects used, OO applications, relationships between classes and objects, class specific data, behavior and physical OOD. Rigidity, Fragility, Immobility and Viscosity are signs of poor design architecture. For Validating quality of OOD Class / Method construct by WMC, Class/Message construct by RFC, Class/Cohesion construct by LCOM, Coupling construct by CBO, inheritance construct by DIT and NOC metrics are used. Some important heuristics are:

- Minimize the number of messages in protocol of a class
- Classes should only exhibit nil or export coupling with other classes.
- Keep related data and behavior at one place.
- Do not create god classes or god objects, because they exhibit non communicating behavior.
- Eliminate irrelevant classes and classes that are outside the system.
- Minimize fan out in a class.
- All data in a base class should be private; do not use protected data.
- All abstract classes must be base classes and vice versa.
- Construct reusable frameworks rather than reusable components.

3. DESCRIPTION AND SELECTION OF DESIGN PATTERN

Design Patterns are described in graphical notation with UML diagrams which capture the end product of design processes. To reuse the design record the designs, alternatives by describing them with pattern name and

classification, intent, motivation, applicability, structures, participants, collaboration, implementation and their uses in real systems. Design Patterns vary in their granularity and level of abstraction. Creational design Patterns concern the process of object creation, Structural patterns deal with composition of classes or objects and Behavioral patterns deals the way in which classes or objects intent and distribute responsibilities. There are several different approaches to find design patterns that are right to solve one design problem. Some of them by considering how design patterns solve design problems, by studying relationship among patterns, by studying patterns of like purpose, by examining cause of redesign and by considering what should be variable in our design.

Creational Design Patterns		Structural Design Patterns		Behavioral Design Patterns	
Class	**Object**	**Class**	**Object**	**Class**	**Object**
Factory Method	Abstract Factory Builder Prototype Singleton	Adapter	Bridge Composite Decorator Façade Flyweight Proxy	Template Method Interpreter	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

4. COMMON CAUSES AND PROPOSED SOLUTION FOR DESIGN PROBLEMS BY USING CREATIONAL

DESIGN PATTERNS

Design patterns solve many problems faced by Object Oriented designers in many different ways. Some common causes of design problems and proposed solutions by using creational, structural and behavioral patterns are tabulated.

Design Problem	Proposed Solution	Design Pattern(s) Used
Creating an object by specifying a class explicitly	Create Objects indirectly	Abstract Factory, Factory Method, Prototype
Dependence on h/w and s/w platform	Design to limit platform dependencies	Abstract Factory
Dependence on object implementation	Change object when needed	Abstract Factory
Algorithmic Dependencies	Algorithms likely to be changed should be isolated	Builder
Tight Coupling	Design with loose coupling	Abstract Factory
Inability to alter classes	Modify a class that cannot be conveniently modified	Adapter
Class with only one instance	Provide global point to access	Singleton
Represent Physical Structure	Represent Part-Whole hierarchies	Composite
Construct Physical Structure	Define family of algorithms	Strategy
Establishing User Interface	Provide sub classing for extending functionality	Decorator
Support multiple look and feel	Portability across h/w and s/w	Abstract Factory
Support user operations	Encapsulate request as an object	Command
Support multiple windows	Decouple an abstraction from its implementation	Bridge, Abstract Factory
Define new operation without	Represent operation of an object	Visitor

changing class	structure	
Convert from one format to many	Separate the construction of complex object from its representation	Builder
Construct a Framework of an application	Define an interface for creating an object	Factory Method
Customize a Framework of an application	Create using prototype instance and create new object by copying	Prototype
Designed for reuse, but not reusable	Convert the interface of a class into another interface	Adapter
Client code platform dependent	Decouple an abstraction from its implementation	Bridge
Build Complex system from its simple component	Compose objects into tree structures	Composite
To add responsibility to an individual object not to an entire class	Sub classing for extending functionality	Decorator
Minimize communication and dependencies between subsystems	Provide a unified interface	Façade
Application does not depend on object identity	Use sharing to support large no of fine grained objects	Flyweight
Create expensive object on demand	Provide a surrogate	Proxy
To avoid coupling	Give multiple objects a chance to handle a request	Chain of Responsibility
To support undo	Encapsulate a request as an object	Command
To represent grammar rule for classes	Use abstract class regular expression	Interpreter
Providing uniform interface for traversing	Create aggregate object	Iterator
To customize specific dependencies	Define an object that encapsulate set of objects	Mediator
To record internal state of an object	Store a snap shot	Memento
Maintain consistency between related objects	One-to-many dependencies	Observer
Design an object that exhibit different behavior in each state	Use an abstract class	State
Different algorithms at different times	Define classes that encapsulate	Strategy
Design an algorithm without changing its structure	Design Concrete classes relies on abstract class	Template Method
Represent an operation without changing the class	Packaging related operations	Visitor
Visit across class hierarchies	Traverse by calling their operations	Visitor
To resolve incompatible interfaces	Convert original interface of a component into another interface	Adapter, Strategy
For better cohesion	Create a pure fabrication object	Abstract Factory
To design with exactly one instance of a class	Define a static method of a class that return a single term.	Singleton
To design for varying, but related algorithm or policies	Define each algorithm/policy in a separate class with a common interface.	Strategy
To treat a composition structure of an object as a non composite object	Define class for composite	Composite
To design a common, unified interface to a set of implementation	Define single point of construct to the subsystem	Façade
To design publisher which wants to	Define a subscriber or listener	Observer

maintain low coupling to subscriber	interface	
Recovery from remote service failure	Location transperancy using service look up	Adapter
Failure to a local service from product information	Add a level of indirection with a surrogate proxy object	Proxy
To create families of related classes that implement a common interface	Define concrete factory class for each family of thing to create.	Abstract Factory
To create persistence objects	Persistence service from a persistence framework	Template Method
To access a persistence service	Provide a unified interface to a subsystem	Façade
To create persistence object does not cause immediate dbase update	Create state class for each state	State
To design a transaction	Make each task a class that implements a common interface	Command
Lazy materialization	Create proxy for another object that meta dbase real submit	Virtual Proxy

Creational design patterns help to make a system independent of how its objects are created, composed and represented. A class creational patterns use inheritance to vary the class that is instantiated, an object creational pattern will delegate instantiation to another object. They encapsulate knowledge about which concrete classes the system uses and how they hide instances which are created. If creational patterns are competitors then either Prototype or Abstract Factory could be used. If they are complementary then Builder can use one of other patterns to implement.

5. STEP BY STEP APPROACH TO USE A RIGHT DESIGN PATTERN TO SOLVE DESIGN PROBLEM

`

Once we have picked a design pattern to solve a design problem apply the following step-by-step procedure effectively:

Step1: Pay particular attention to the applicability and consequences of selected design pattern(s) to solve design problem.
2: Understand the classes and objects in the pattern and relationship among them by knowing their participants, structure, and collaboration.
3: Implement the pattern with specific code.
4: Name Pattern Participants used in solving design problem.
5: Identify existing classes that pattern will affect and modify them accordingly.
6: Use responsibilities and collaborations associated with each operation involved.
7:Implement the operations to carry out the responsibilities and collaborations in the selected pattern(s) to solve the design problem

6. STUDY OF UTILIZATION OF CREATIONAL DESIGN PATTERNS IN SOLVING DESIGN PROBLEMS

Factory method or Visual constructor lets a class defer instantiation to subclasses and define an interface for creating an object and to construct a frame work for an application. While implementing two aspects to be considered are when creator class is an abstract class and when is a concrete class. Templates can used to avoid sub classing.
Abstract Factory or Kit provide an interface for creating families of related objects without specifying concrete classes , to create objects indirectly to implement loose coupling, to create pure fabrication object. When an application needs only one instance it can be implemented as Singleton.
Builder, separate the construction of a complex objects from its representation and to convert from one format to another. An abstract builder class defines an operation for each component, which overrides operations for components.
Prototype, specify the kinds of objects to create using prototypical instance and to create a new object by copying this prototype. It will be implemented by using a prototype message or by clone operation.
Singleton, ensure a class only has one instance, and to provide global point of access. Design starts by using Factory Method and evolve towards other creational patterns. Designs that use Abstract Factory, Prototype and Builder are more flexible than those of Factory Method.

1. Consider user interface that support multiple look and feel standards for Fund Transfer design problem. Different look and feel define different appearances and behaviors for this user interface Fund Transfer like through ATM counters, Bank counters and Net banking. Design to be portable

CENTRE OF PROFESSIONAL RESEARCH PUBLICATIONS
ISSN 2049-3444
International Journal of Engineering and Technology Volume 1 No. 3, December, 2011

across look and feel standards we solve this problem by defining and Abstract Fund Transfer Factory design pattern that declares an interface for creating each basic kind, which represented with design tool in fig.1a and another case for Interviewing technique for a particular candidate, represented in fig.1b and a user interface tool kit that support portability, is solved by defining an abstract factory design pattern represented in fig 1c.

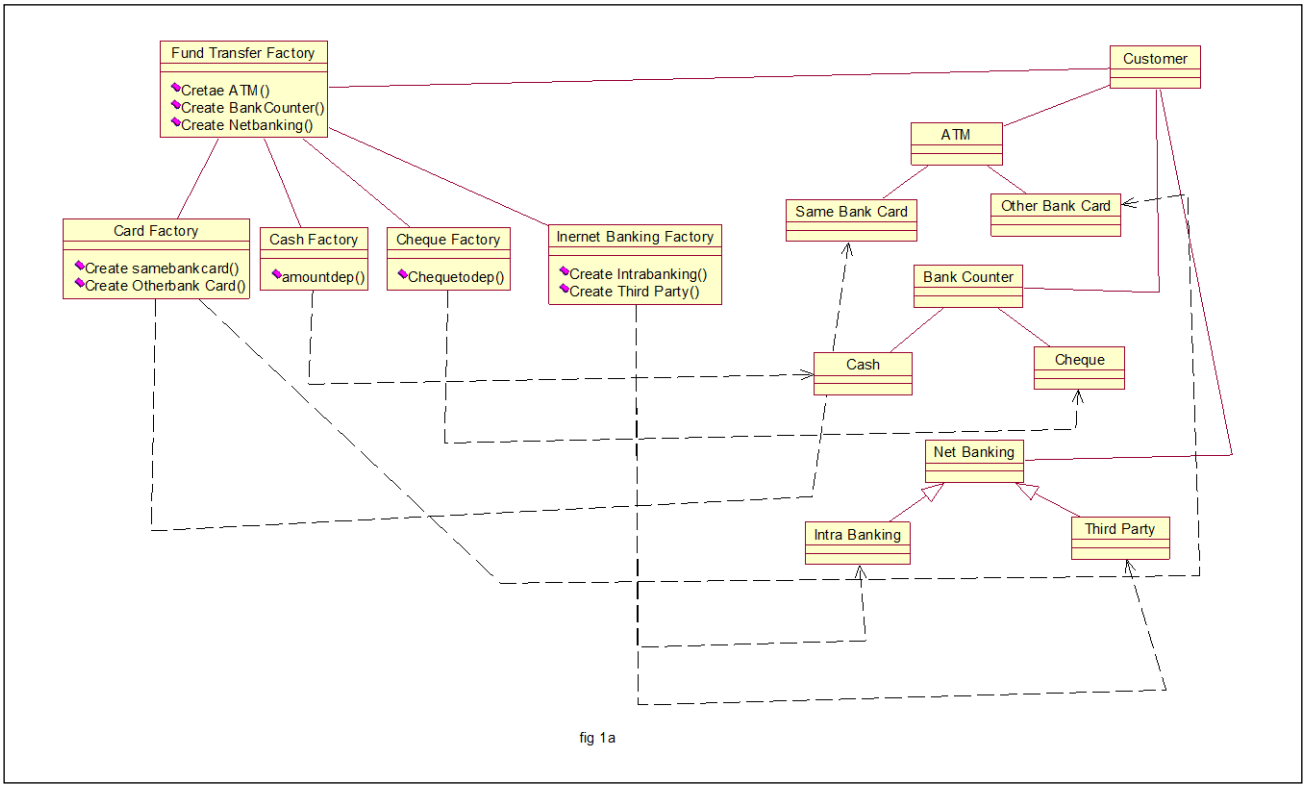


fig 1a

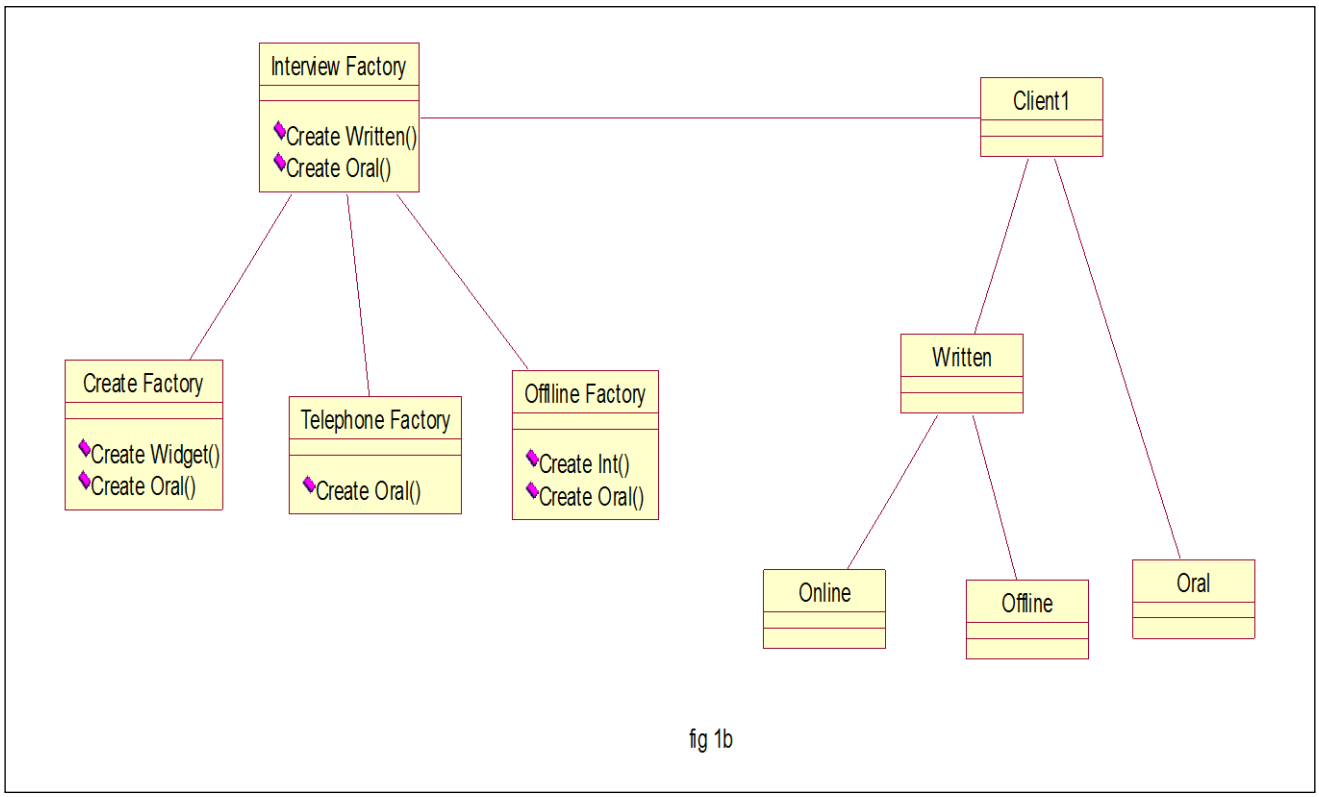


fig 1b

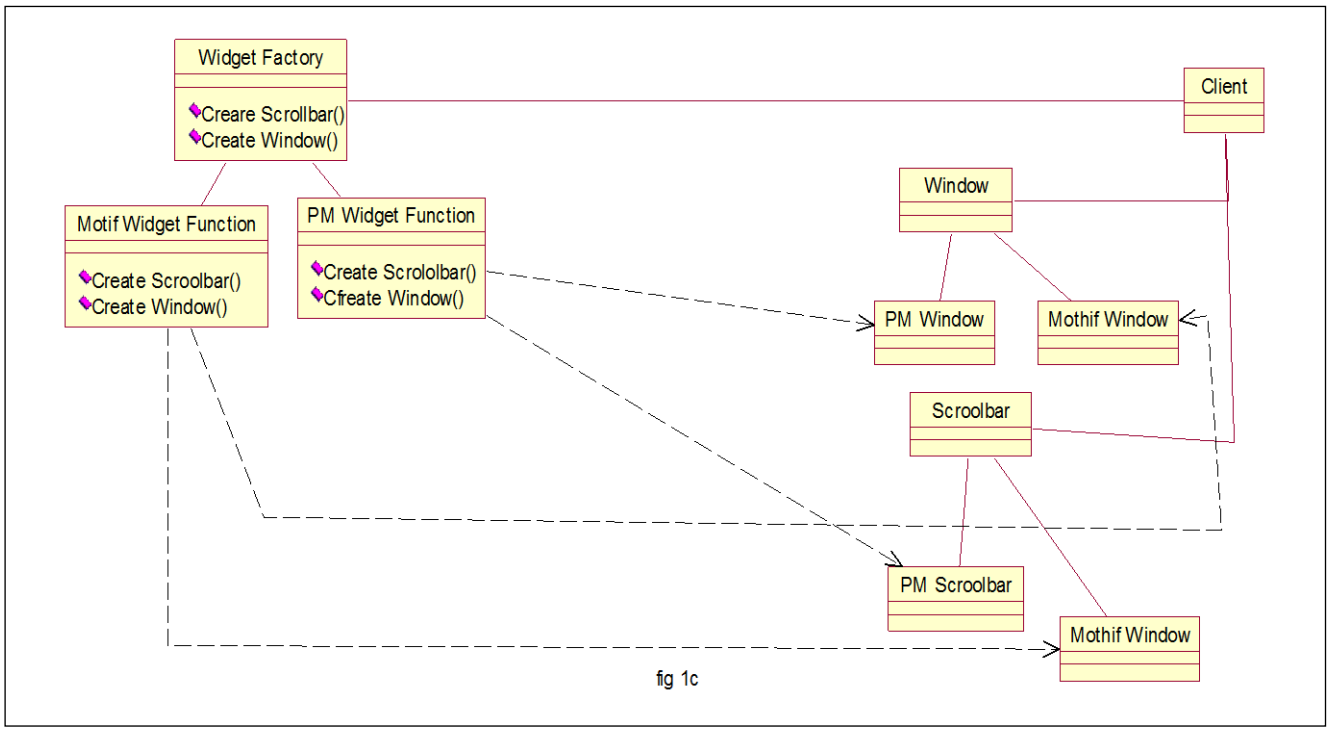


fig 1c

2. Consider a Parse Text document to convert into different formats that can be edited interactively. Problem is the number of possible conversions should be easy to add a new conversion without modifying the reader. Solution to this problem is by defining Builder design pattern to configure PTF Reader with a text command object, which is represented in fig.2a and another case of Traversal technique of a Binary Tree represented in fig. 2b.

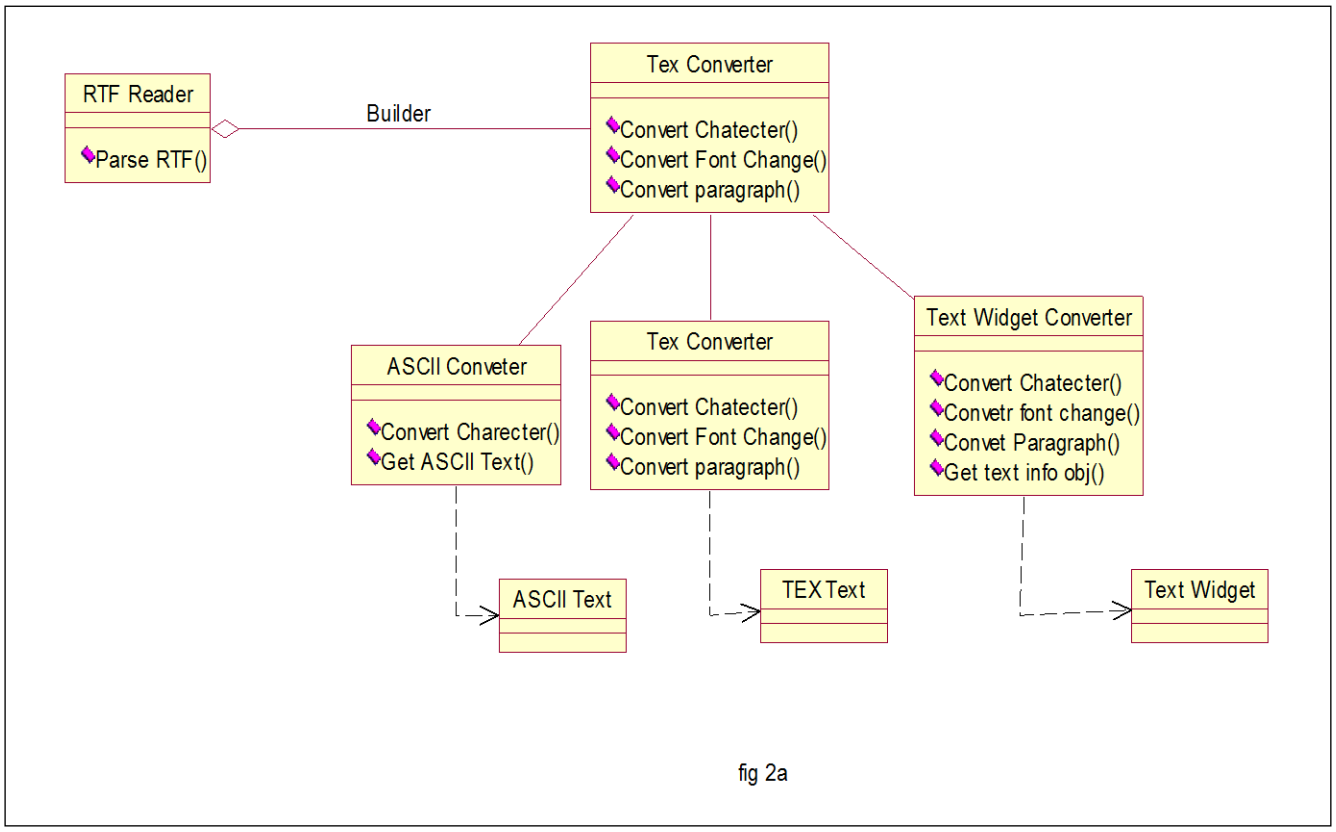


fig 2a

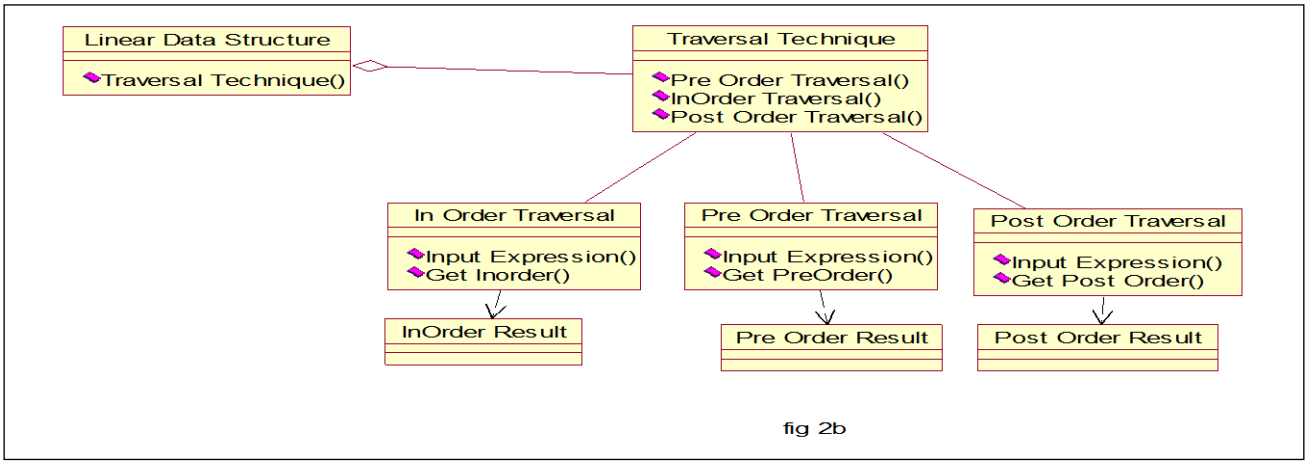


fig 2b

3. Consider a framework for application that can present multiple transactions (local and global) to the user. Two abstractions in this framework are the Class Application and Transaction. The Factory Method design pattern offers a solution to this design problem. It encapsulates the knowledge of which transaction subclasses to create and moves this knowledge out of the framework, which is represented with the help of tool design in fig.3.

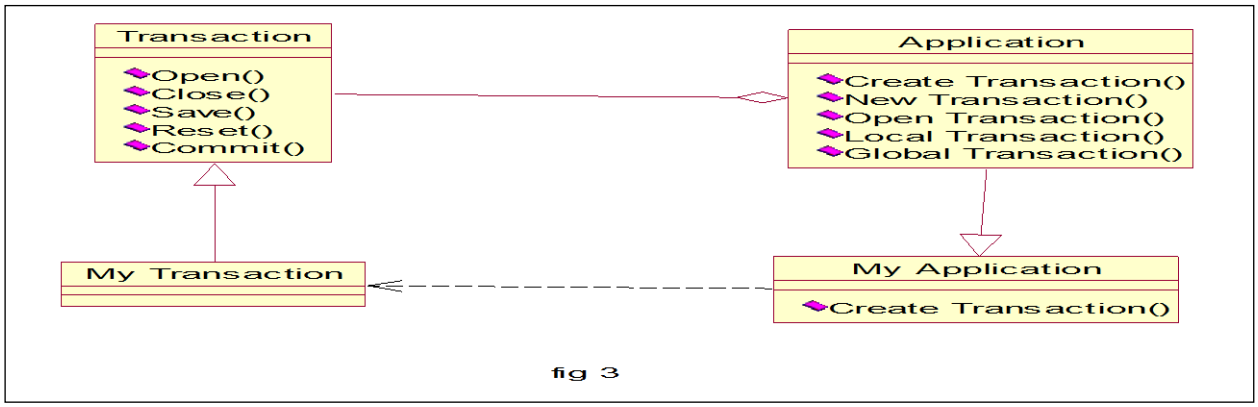


fig 3

4. Consider to build an editor for music scores by customerizing a general framework for Graphical editors and adding new objects notes, rests and staves. The solution for this design problem lies in making graphic tool create a new graphic by copying an instance called Prototype design pattern represented in fig.4a by using design tool.

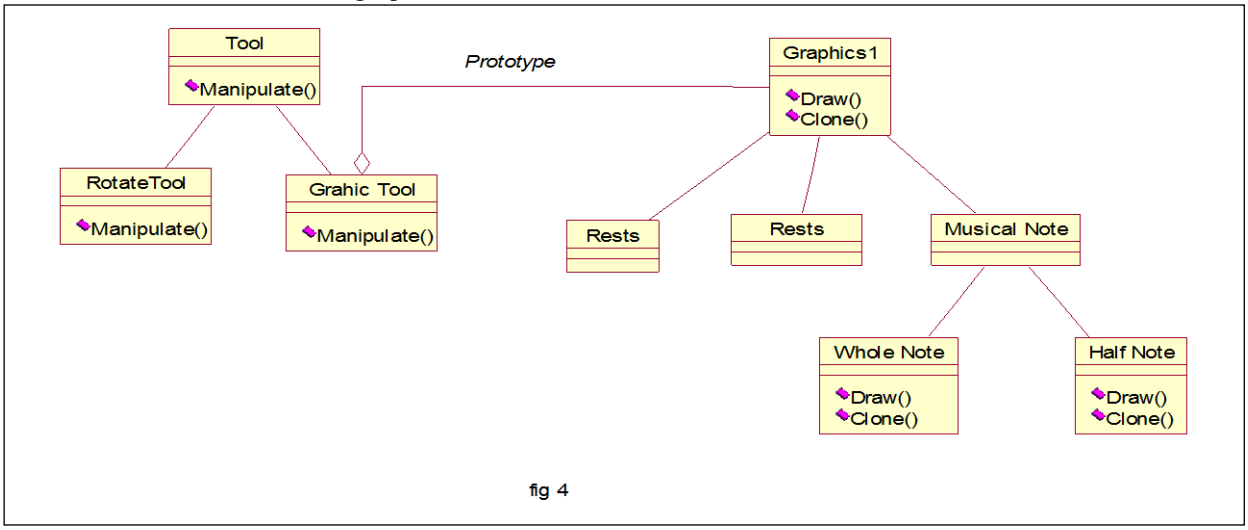


fig 4

International Journal of Engineering & Technology
CENTRE OF PROFESSIONAL RESEARCH PUBLICATIONS
ISSN 2049-3444
International Journal of Engineering and Technology Volume 1 No. 3, December, 2011

5. Consider some classes to have exactly on instance for example Many printers in a system, only one Printer Spooler and another case one file system with one window manager. Such types of design problems will be solved with the help of Singleton design pattern which is represented in fig. 5a and 5b.

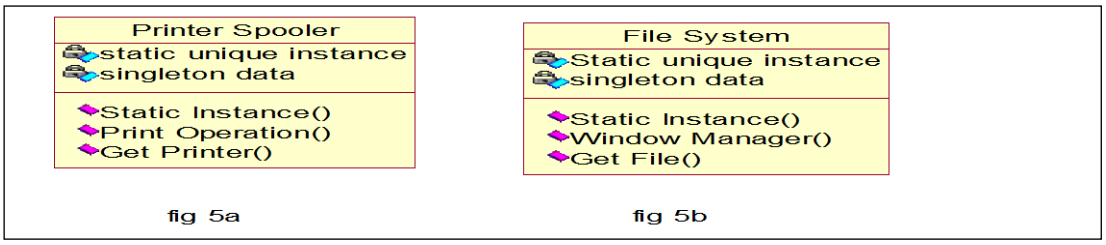
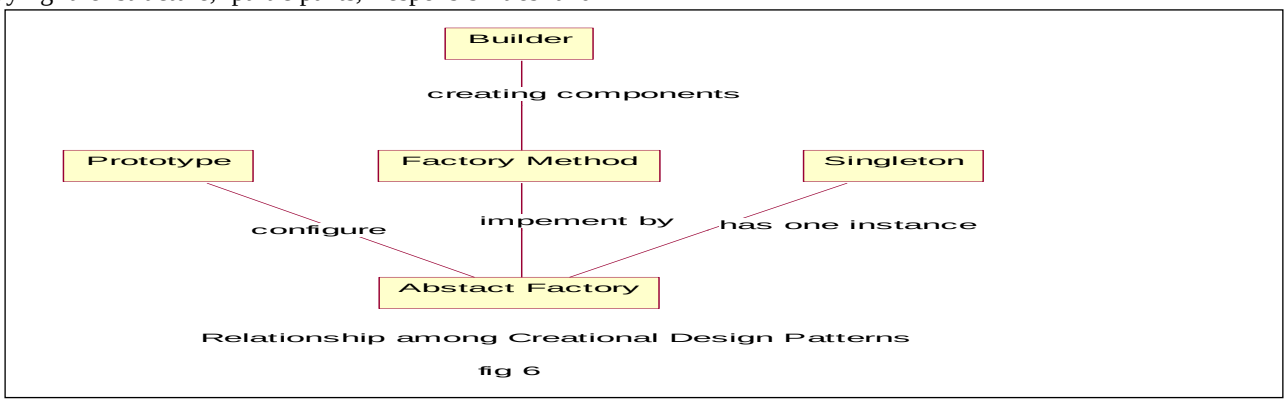


fig 5a fig 5b

7. RELATIONSHIPS AMONG CREATIONAL DESIGN PATTERNS

After selecting design pattern(s) to solve design problem we have to understand classes and objects participates by studying the structure, participants, responsibilities and collaborations of each pattern and relationships existing among them. Relationship among Creational design patterns represented in the fig 6. In fig 7 relationship among structural components of Creational design patterns and in fig.8 relationship among creational design patterns for Presentation Manager are shown.



Relationship among Creational Design Patterns

fig 6

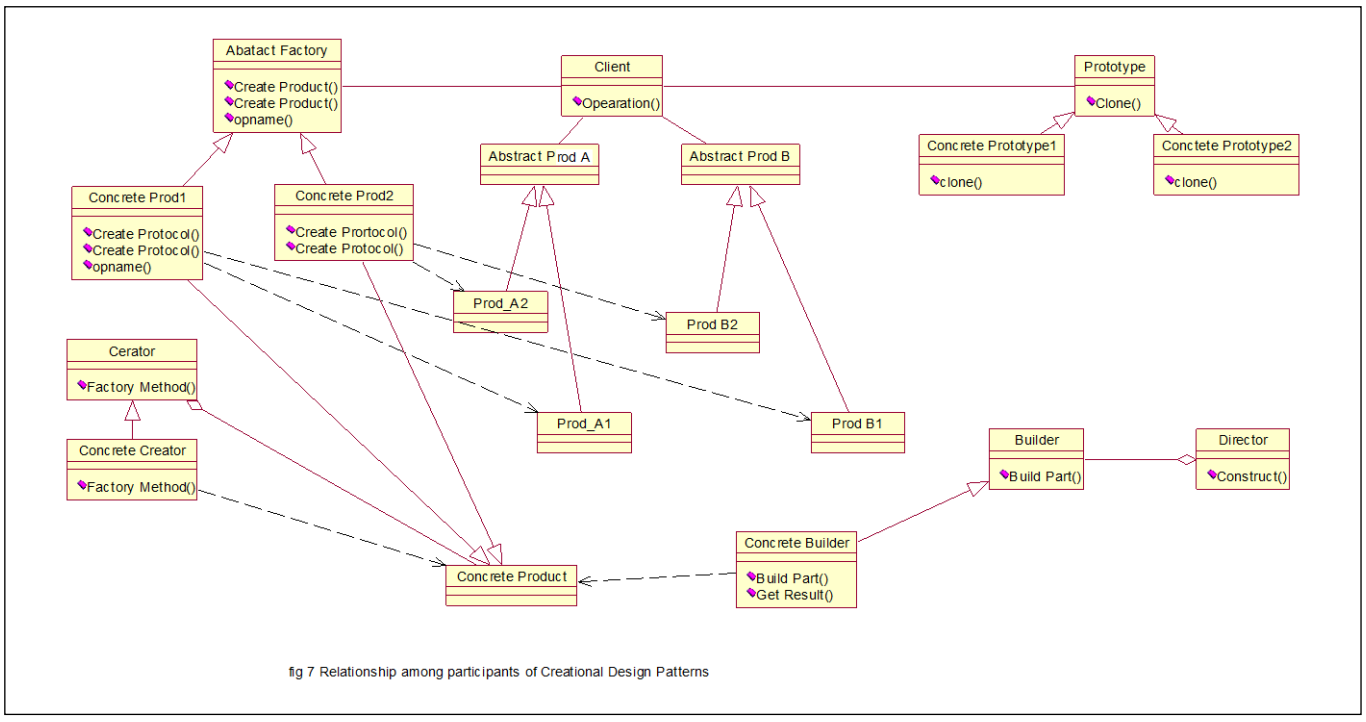


fig 7 Relationship among participants of Creational Design Patterns

International Journal of Engineering & Technology
CENTRE OF PROFESSIONAL RESEARCH PUBLICATIONS
ISSN 2049-3444
International Journal of Engineering and Technology Volume 1 No. 3, December, 2011

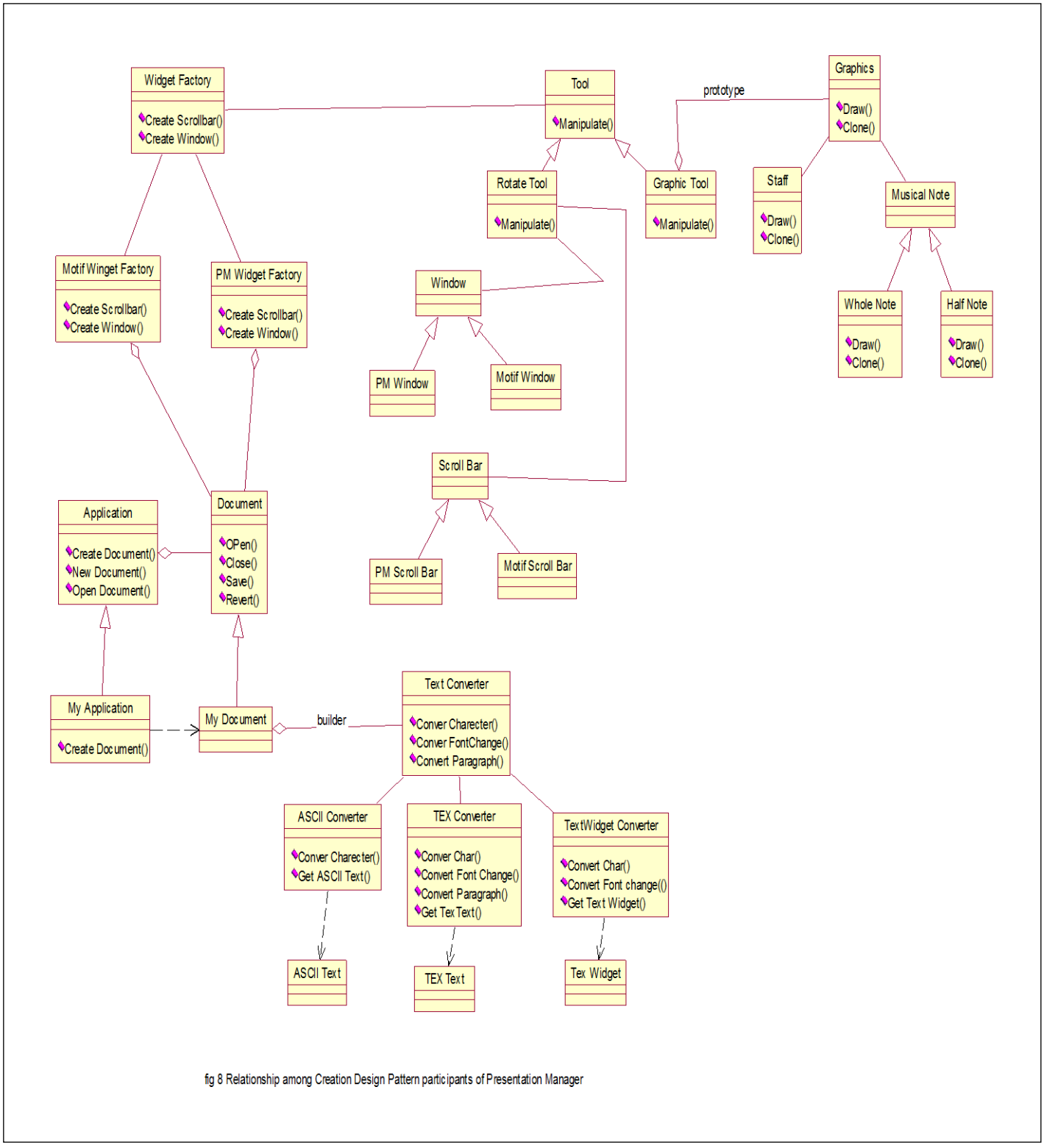


fig 8 Relationship among Creation Design Pattern participants of Presentation Manager

8. CONCLUSION

The purpose of this paper is to study utilization of Creational Design Patterns in solving design problems of Object Oriented Design. Some of them are identified and proposed solution suggested with relevant design patterns. Some case studies in designing real systems are considered with some design problems and solutions proposed by using Creational design patterns are represented with UML diagrams by using Rational Rose software. Relationships among Creational design patterns, relationships among

structural components of Creational design patterns presented.

REFERENCES

[1] European Conference on Object Oriented Programming, pages 21-35, Kaiserlautern, Germany, July1993.

[2] European Conference on Object Oriented Programming, pages 139-149, Bologna, Italy, July1994.

[3] Object Oriented Analysis and Design with Applications, Benjamin/Cummings, 1994, Second Edition.

[4] ACM Transactions on Programming Languages and Systems, pages 343-387, October 1981.

[5] An Object Oriented System in C++ Communications of the ACM, pages 117-126, September 1993.

[6] ACM User Interface Software Technologies Conference, pages 92-101, Snowbird, UT, October 1990.

[7] Software Reusability, Volume II: Applications and Experience, pages 269-287, Reading MA, 1989.

[8] Object Oriented Programming Systems, Languages and Applications Conferences Proceedings, pages 214-223, Portland, November 1986.

[9] Proceedings of the USENIXC++ Conference, pages 51-63, Washington, April 1991.

[10] Object Oriented Design Heuristics by Reil.

[11] Object Models by Coad.

[12] Designing Object Oriented Software by Brock.

[13] Using Pattern Languages for Object Oriented Programs, AFIPS Conference Proceedings, Pages 43-47, Beck K and Cunningham W 1987.

[14] Introduction and overview of the Multics Systems AFIPS Conference Proceedings 27, pages 185 – 196, Carbato F and Vyssotesky V

[15] The Unified Modeling Language Reference Manual 2e, Addison-Wesley, Rumbaugh J , Jacobson and Booch G 2004.