

CORBA Design Patterns In Distributed Systems

Nothmar Noriel

Department of Computer Science and Information Systems, PACE University

Abstract

Design Patterns in CORBA are effective reusable solutions to many problems that result in applying CORBA for the interoperability of applications in a heterogeneous system. This paper will discuss an overview of some Design Patterns commonly used in applying CORBA Architecture to implement Distributed Systems.

This paper will assume that the reader has some knowledge of distributed systems.

1 Introduction

The Computer Revolution of the late 1900s, technological advances of computer systems have resulted in the evolution of many different competing computer architectures. These different computer systems have produced scores of computer applications developed to work specifically for that specific operating system. In today's global economy, interaction of computer applications in a Distributed System is of high value and importance. Allowing one computer application to update information in another application located in a different computer system is just one example of the need for a Distributed system that manages these resources.

Distributed Systems is a network of various heterogeneous computers and processors. The difference of Distributed systems and Network systems are that Distributed Systems allow a uniform view of naming schemes, file systems, and access to resources, while Network systems does not hide the fact that system resources that are used belong to another computer system.

The Common Object Request Broker Architecture (CORBA) is an effective resource in allowing interoperability of heterogeneous systems in a Distributed System. CORBA is emerging as a standard for distributed computing and has a lot of advantages that make use of distributed computing.

Design Patterns are reusable solutions that assist in more effective development of computer systems. Design Patterns are collected solutions to recurring problems that can be resolved in a similar manner with a possible similar solution. Identification of these type of recurring problems is critical in applying Design Patterns to resolve them.

This paper will present an overview of CORBA Architecture, an overview Design Patterns and then discuss some interesting CORBA Design Patterns.

2 Distributed Systems Overview

Distributed systems are groups of computers and processors connected by a network. Distributed systems allow a collection of computers to provide a similar global view of shared resources.

Distributed Systems vs. Network systems

The difference of Distributed systems and network systems is that network systems do not attempt to provide a consistent global view of shared resources. Distributed systems are tightly coupled systems and network systems are not tightly coupled. Distributed systems are tightly coupled because distributed systems have many rules that must be followed in order to maintain integrity.

Distributed Systems Models

Distributed Systems have two primary models: Client Server model and Peer-to-peer model. The Client Server model has a client node making a request and a Server node receives the request and processes the request for the clients. In a peer-to-peer model, every computer in the distributed system can process the request of another computer.

Transparency

Distributed systems provide various levels of transparencies to each node in the network. A distributed system provides name transparency, location transparency, access transparency, migration transparency, replication transparency, concurrent and parallelism transparency and failure transparency. Transparency mainly means that when a user requests a service, the user will not be able to distinguish whether the shared resource resides on his current PC, or from another PC system in the distributed system.

3 CORBA Overview

CORBA was designed by the Object Management Group (OMG) to primarily provide an object-oriented interoperability of applications in heterogeneous distributed system. The use of Object-Oriented design, analysis, and development using CORBA allows greater reusability across systems. Advantages of Object-Oriented features such as inheritance, encapsulation, redefinition and dynamic binding are implemented in CORBA. Object-oriented features of CORBA objects are also effectively easier to extend and modify without affecting other applications and objects.

CORBA encapsulates applications and provides a common infrastructure for communication using the CORBA ORB and is used to receive requests and locate server objects.

CORBA encourages the development of open applications, applications that can be integrated to larger systems.

CORBA advantages also include: location transparency, programming language transparency, Operating System transparency and Computer Hardware transparency.

Location transparency

CORBA Objects can be located and accessed in the client or reside on a remote server. The location of the CORBA Objects should not affect its implementation ability or not system performance.

Programming Language transparency

CORBA supports many different languages including Java, C/C++, Ada 95, Smalltalk and COBOL.

Operating System transparency

CORBA supports many O/S platforms including Microsoft Windows families: NT/2000/XP, various UNIX flavors, Mainframe O/S.

Computer Hardware transparency

CORBA objects can be used on different Hardware configurations and Architectures

CORBA Architecture

CORBA architecture can be categorized into four software interfaces: CORBA Object Request Broker (ORB), CORBAservices, CORBAfacilities and CORBAdomains.

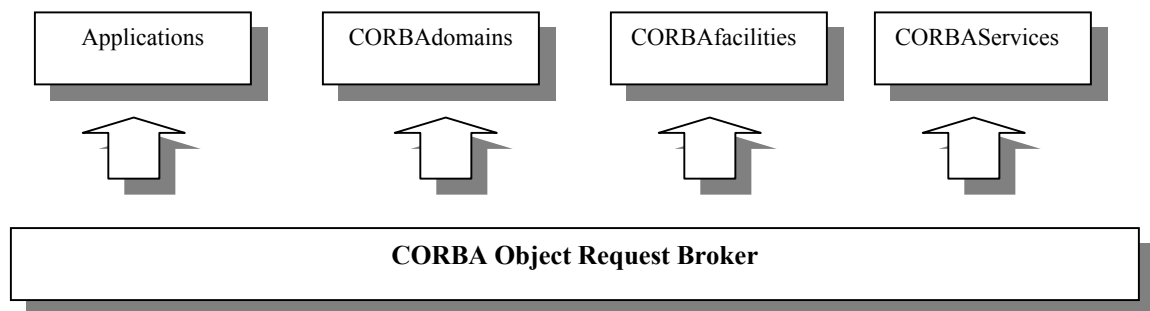


Figure 1: CORBA Architecture

CORBA Object Request Broker (ORB)

The heart of CORBA is the Object Request Brokers (ORB). CORBA ORB is interfaces that allow a client to access distributed objects residing on servers.

A request by a client for an object goes to the ORB. The ORB locates the requested object and identifies the server that hosts the requested object. The ORB translates the client and server calls. The ORB then maps the Interface Definition Language (IDL) to the language to which the object is implemented.

CORBAdomains

CORBA domains are vertical markets, including Brokerage, Financial and Pharmaceutical industries where CORBA is applied

CORBAfacilities

CORBA Facilities are horizontal interfaces focused on Application interoperability in distributed systems

CORBA Services

CORBA Services include object life cycles, event notification, and distributed system transactions

CORBA IDL

Servers use the CORBA Interface Definition Language (IDL) as a generic interface to Objects that it hosts. The interface includes implementation information such as attribute and operation signatures.

CORBA Object Adapter

The CORBA Object adapter is mainly responsible for creation and translation of CORBA object references, activation and deactivation of Object implementations and mapping object references with object implementations.

4 Design Patterns Overview

Design Patterns are reusable solutions to recurring design problems. Design Patterns can be thought of as a solutions template to a pattern problem. The resurgence of Design Patterns can be attributed to the Gang of Four: Gamma, Helm, Johnson and Vlissades. These four authors authored the industry popular book: Design Patterns: Elements of Reusable Object-Oriented Software.

Patterns can be described using the following information: Pattern Name, Pattern Intent, Solution, Solution Benefits and Consequences, References to other Design Patterns, Examples, Forces

Pattern Name

A Unique Name that identifies the Pattern

Pattern Intent

The purpose of the Design Pattern

Solution

Description of how the Design Pattern solves the problem

Solution Benefits and Consequences

Describes the Benefits and Consequences of using the Design Pattern

References to other Design Patterns and Solutions

Describes other Similar Design Patterns and Solutions

Examples

Gives examples of the Design Pattern solution

Forces

Forces are the motivating factors that have a need to be resolved. Forces include the environment and information of the problem scope. Design Patterns attempt to provide solutions to these forces.

Some Essential Design Patterns that I will discuss include the Singleton Pattern, Façade Design Pattern, Adapter Design Pattern and Abstract Factory Pattern. Other Essential Design Patterns, that I will not discuss in this research paper are: MVC Pattern, Bridge Pattern, Strategy Pattern, Observer Pattern, Template Method Pattern.

Singleton Pattern

The Singleton Pattern ensures that an object is only instantiated once. A method is invoked, this method checks if the specified object has already been created. If the object has only been created, the reference to this object is returned by the method. If the object has not yet been instantiated, then the object is created and a reference to the objects is returned by the method.

Singleton Pattern is for client objects that need to call the same object, and not want to create another object.

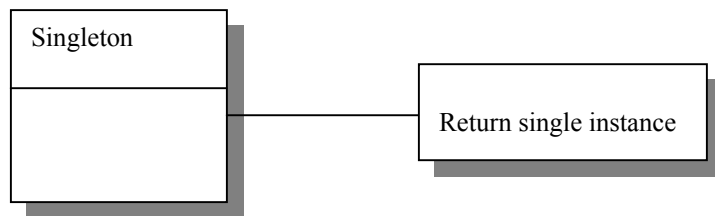


Figure 2: Singleton Pattern

Façade Design Pattern

The Façade Design Pattern provides a new interface in front of the applicable system. The Façade Design Pattern can be used to simplify a complex interface. The Façade encapsulates the original system and provides its own interface. Benefits include: The Façade can include new functionality not previously offered by the original system and consequences are that some of the original systems functionality may be lost.

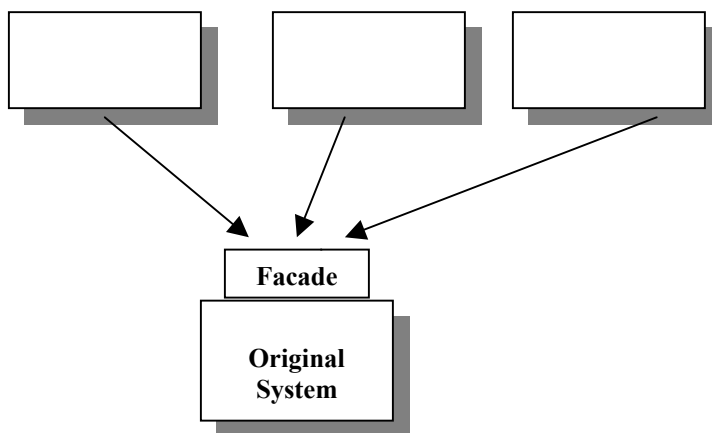


Figure 3: Façade Design Pattern

Adapter Design Pattern

The Adapter Design Patterns replicates the plug adapter, in that it converts the current systems interface into another interface by wrapping the entire interface and creating the desired new interface. The benefits of this pattern are that it allows objects to be encapsulated by a new class structure and creates new interfaces that match the class that invokes it.

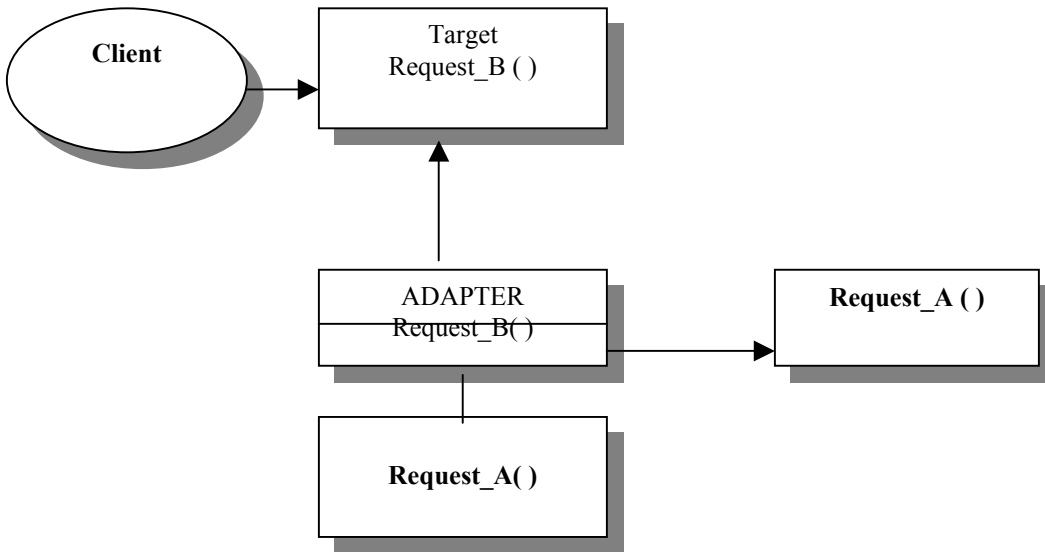


Figure 4: Adapter Design Pattern

Abstract Factory Pattern

The Abstract Factory Pattern is a useful pattern that collectively creates families of objects. The Abstract Factory manages the rules in instantiating these objects. Client objects invoke factory objects to instantiate the required server objects. When the factory object is invoked, an Abstract class is created. The Abstract class contains an interface that provides method invocation for the objects that are instantiated. The consequence of this pattern is that rules on which objects to use are isolated from the logic on how to use the objects. Modern compiler designs make use of the Abstract Factory Pattern to instantiate its objects.

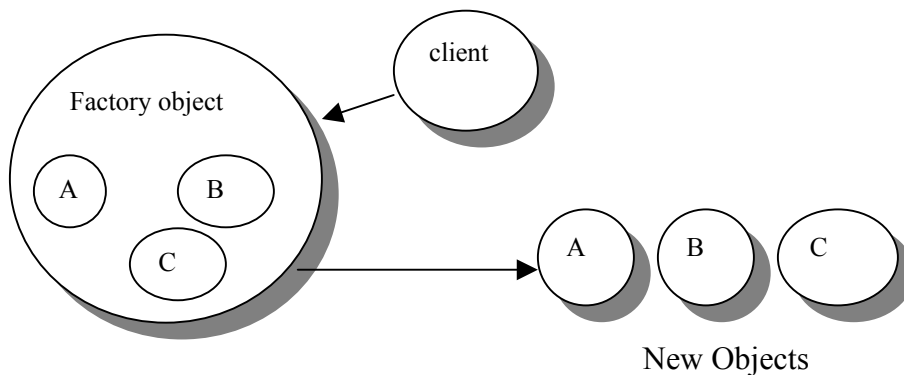


Figure 5: Factory Design Pattern

5 CORBA Design Patterns in Distributed Systems

CORBA Design Patterns are Design Patterns that resolve recurring problems in CORBA implementations in Distributed Systems. CORBA Distributed systems provide the following services: Partial Processing, Concurrency control and Mutual Exclusion. Design Patterns discussed here replicate many existing features implemented by Distributed Systems. CORBA facilitates distributed systems.

Parallel Systems

Parallel Systems are systems with multiple processors located within the same computer. Parallel systems can perform parallel processing of processes and jobs by utilizing different processors that are connection by various network architectures including Interconnection Network, Bus Network, Crossbar network, Hypercube network and Shuffle-Exchange Interconnection network.

Concurrency Control

Concurrency control is an essential operating system concept that is crucial also to Distributed systems. Concurrency control manages processes from accessing a shared resource and also controls efficient processing of distributed and parallel systems.

Mutual Exclusion

Mutual Exclusion prevents multiple processes from accessing shared resources at the same time. The shared resource that is currently being accessed is referred as the critical region.

Design Patterns for CORBA in Distributed Systems that I will discuss are: CORBA ORB, Partial Processing, Lock, Replication, Naming Service, Load Balancing and Concurrency Control.

Design Pattern: CORBA Object Request Broker (ORB)

The Object Request Broker is an enterprise-wide CORBA Design Pattern that simplifies the overall Distributed System. The ORB decreases the complexity of a system by providing client invocation of server objects that provide transparency of low-level implementations. The ORB acts as the middleman (middleware) for communication of the client and server.

Advantages of the CORBA ORB

ORB relieves the application from performing infrastructure functions such as data marshalling, server location/selection/activation. Client requests are provided with the information requested without regard to how the information was processed.

Disadvantages of the CORBA ORB

Applications vendor's interfaces to the CORBA standard may result in dependency to CORBA implementations, which is against the low coupling goal of object-oriented systems

Design Pattern: Partial Processing

Partial Processing increases performance by reducing wait time on object creation by factory objects. The server creates minimal attributes for the object, including object reference and minimal data to allow the creation of the object after invoking the object with the object reference. Performance management is the motivating force for partial processing.

Advantages of Partial Processing

Partial processing increases parallel processing in distributed systems. Partial processing reduces processing time for jobs waiting for results.

Disadvantages of Partial Processing

Major consequence of partial processing is there is more complex maintenance in implementing Partial Processing

Design Pattern: Lock

Lock is used to prevent multiple processes from accessing a shared resource. The Lock can be either locked or unlocked. Lock manages concurrency control and mutual exclusion to shared resources. In order for a process to use a shared resource, the shared resource should be in the unlocked state.

Advantages of Lock

Lock manages concurrency control in distributed systems, protecting shared resources and allows parallel processing

Disadvantages of Lock

Lock can reduce system performance, especially for a critical region that is accessed repeatedly.

Design Pattern: Instance Reference

The Instance Reference Design Pattern creates a mapping of an object interface with the requested object instance. An object can be referenced by an identifier reference data specified by the object implementation that is created prior to the creation of the object reference. An operation that requests the use of an object instance calls this object reference.

Advantages of Instance Reference

This Design Pattern increases performance by creating multiple object instances for each server process. Object Reference allows the mapping to an object reference.

Disadvantages of Instant Reference

Implementation of Instance Reference increases code complexity in development. Instant References may require specific standard implementations in order to reference objects

Design Pattern: Replication

Replication Design Pattern is popular in replicating data objects and information to various locations. Motivating forces for the Replication Pattern is the need to provide backup of data objects and information.

Advantages of Replication

Replication increases Performance because it can create multiple objects to handle processing and sharing of data from many locations of a distributed system. Replications are cheaper than copying data.

Disadvantages of Replication

Replication can become costly for resources and can become complex by trying to coordinate the consistency of data in various locations.

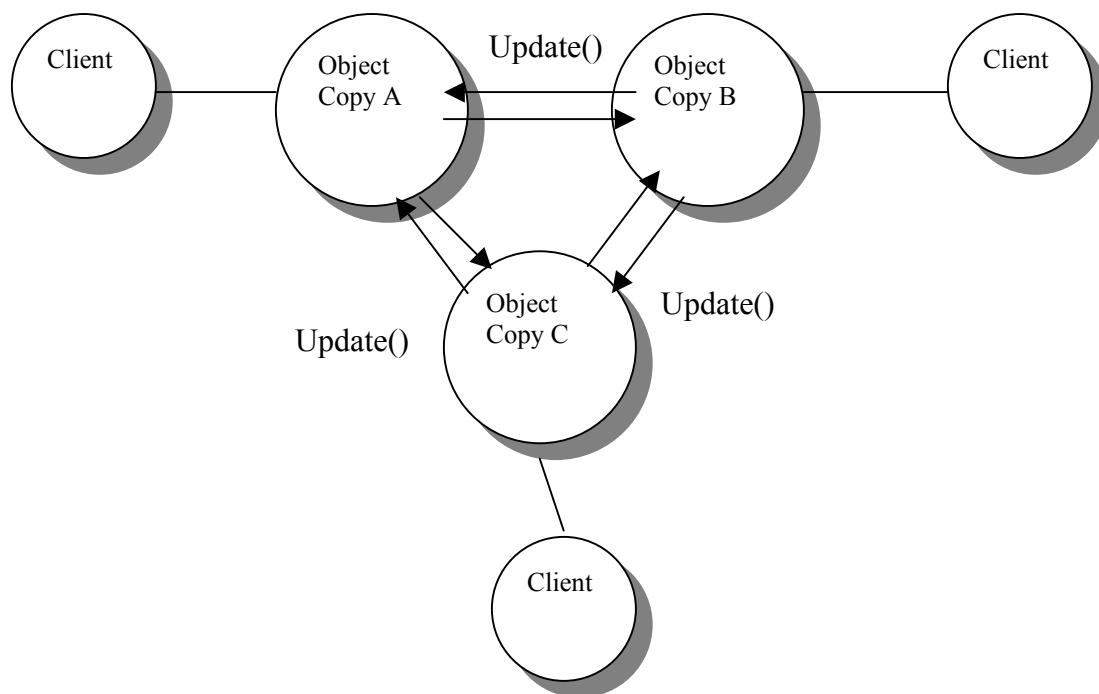


Figure 6: Replication Design Pattern

Design Pattern: Naming Service

Naming is a repository that store name object references. The Naming service is used to located objects in CORBA. The Naming service can be used to retrieve the object reference. Operations of the service can be invoked, through the object reference. Naming supports two basic operations: store (bind) and retrieve (resolve). Services advertise their availability to the Naming Service by their name object references.

Advantages of Naming Service

Naming supports distributed environments. Naming Service is a standard service and implementation that is platform independent.

Disadvantages of Naming Service

Naming Services is not available for all implementations, but can be easily implemented if required

Load Balancing Design Pattern

This Pattern increases the performance of the Distributed System by spreading the work of a server to other servers in the Distributed Systems. Load balancing's main interest is to get the best results in conjunction with effectively utilizing system resources. To perform Load balancing, processes are migrated to available servers. Process will be completely migrated as a whole to another server if the process is indivisible. If the process is divisible, the process can be broken up and parts of the process are migrated. If the process is divisible, there must be effective management in achieving completion of the divisible of the process.

Design Pattern: Object Wrapper

The Object Wrapper pattern is an abstraction that gives an object-oriented interface to applications that lack an OMG IDL interface. The Object Wrapper interface that is created for the application must expose only the required attributes and operations that are required by CORBA clients and services.

Advantages of Object Wrapper

Object Wrappers are especially useful for Legacy applications that lack Object-oriented capabilities and an OMG IDL interface. Object Wrappers make use of object-oriented features without changing the underlying application.

Disadvantages of Object Wrapper

To implement an Object Wrapper for an application, thorough knowledge is required of the application to be interfaced. The operations of the application are then mapped to the object wrapper. Also, limitations of the legacy systems may be inherited when implementing object wrappers.

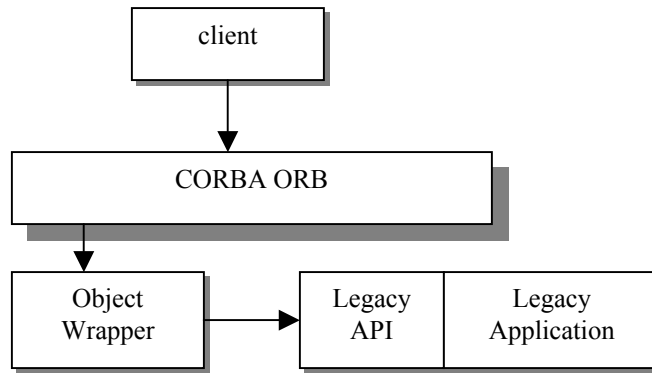


Figure 7: Object Wrapper

6 Conclusions

CORBA Design Patterns provides: [1] Performance Increase, [2] Complexity Simplifications. CORBA provides location transparency, programming language transparency, Operating System transparency and Computer Hardware transparency.

Design Patterns are reusable solutions to recurring design problems. Design Patterns can be thought of as a solutions template to a pattern problem.

CORBA Design Patterns include Parallel Processing, Lock, Replication, Load Balancing, Object Reference, ORB.

In this paper, I provided an overview of various CORBA Design Pattern as well as give an overview of CORBA Architecture and Design Patterns.

References

1. T. Mowbray, R Malveau, *Corba Design Patterns*, John Wiley and Sons, Inc, Canada, 1997
2. D. Galli, *Distributed Operating Systems*, Prentice Hall, NJ, 2000
3. A. Shalloway, J. Trott, *Design Patterns Explained*, Addison-Wesley, NJ, June 2001
4. P. Lewis, A. Bernstrin, M. Kifer, *Databases and Transaction Processing*, Addison-Wesley, 2002
5. R. Pressman, *Software Engineering, A Practitioner's approach*, McGraw Hill, NY, 2001
6. S. Baker, *Corba Distributed Objects using Orbix*, Addison-Wesley, ACM, 1997