Half-Push/Half-Polling

Youngsu Son[1], Jin-Ho Jang[2], Jemin Jeon[2], Sangwon Ko[2],Hyuk-Joon Lee[2], Jungsun Kim[2]

[1] Home Solution Group,
Korea Headquarter,
Samsung Electronics
arload.son@samsung.com

[2] High-Performance Computing and Object-Oriented Technology lab.,
Department of Computer Science at Hanyang University,
1271 Sa 1-dong, Sangok-gu, Ansan, Korea,
http://hpclab.hanyang.ac.kr/

Abstract

To create constantly evolving software, Upgrading became an essential factor. There are two ways to upgrade, those are pushing and polling. Polling has advantage of keeping lastest version of all clients but can cause heavy load of server by simultaneous connecting of clients and unnecessary network traffic. On the other hand, push cause much less load because can upgrade specific client by choosing but there is cumbrous monitoring to keep stopped clients lastest version. I suggest Half-Push/Half-Polling pattern mixing these two different ways, keeping advantages, eliminating disadvantages.

1. Introduction

Software must reflect the real world. If not, software decays over time. No matter how well-made, software will always be revised based on client's requirements and the need for new services which also change every moment. Due to that, many applications currently support upgrade services for clients in order to improve customer satisfaction. Through this upgrade service, clients can use the latest service without installing program every time.

One of the major issues that those companies consider for the Upgrade service is the data transmission method. In the majority of cases, server resource is limited. For this reason, an efficient data transmission method is needed.

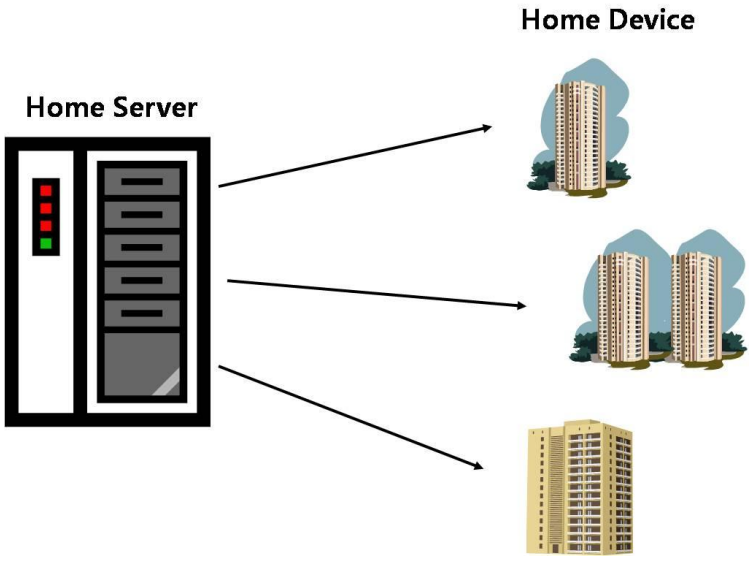
In the client/server model, polling and pushing are generally used as data transmission methods. However, the polling method can cause server overhead when many clients request the

upgrades simultaneously. And the push method has problems such as failure to complete the upgrade when the client is offline or errors occur while upgrading. So the clients who can't do the upgrade must be managed continuously.

A more efficient data transmission method, therefore, is needed to make up for these weak points. In this paper, we present the Half-push/Half-polling pattern which complements the weak points of polling and push methods. The pattern is for a data transmission method which reduces server overhead as well as applies suitable upgrade considering the feature of each client using distributed upgrade time and scheduling.

2. Example

Suppose we are developing a Home Network system for continuous upgrade of a home device version over the network. We adopt the polling upgrade method as data transmission for checking the newest version connected to Home Server. When the home device boots, however, after system starts service in a few days, the server goes down by requesting the upgrades simultaneously from many clients. In addition, the system needs to support different upgrades according to different security policy on each owner of the home network system.



3. Context

A software system designed to support the data transmission of upgrades reducing server overhead and an ongoing upgrade service.

4. Problem

Imagine we are building the system for data transmission to service the newest version continuously to clients. In the client/server model, polling and push methods are generally used data transmission methods for limited server resource. However, the polling method can cause server and client overhead when many clients request the upgrades simultaneously and check server version by periods. Moreover, all clients must retain the same version.

If we adopt push method on the system, we can get the benefit from reducing the overhead using the scheduling. But push method has problems when the client cannot complete the upgrade, for example when the client is off-line or it encounters an error during upgrading.

We, therefore, need efficient data transmission method to maintain the condition of clients using scheduling. The following items should be regarded as forces.

Force

- Consider that the server has the throughput to make upgrade possible.
- Can be possible to upgrade for selected clients.
- The clients who were failed during the upgrade can upgrade in future.
- Various kinds of clients can be upgraded by the system.

5. Solution.

The alf-push/half-polling pattern overcomes the disadvantages of upgrading based on either push and polling method in previous chapter.

The Pusher, in the upgrade server, distributes the upgrade time to pollers(clients) which are on the upgrade list and on-line system using Scheduler. At the distributed time, the client requests upgrade from the pusher which then excutes the upgrade service. (Is the "upgrade service" on the server or the client? Therefore we can avoid the Non-Stop Talker which polling method invokes. Normally to overcome the disadvantage of push method, the clients who are off-line have to be managed additionally. But in our pattern, when those clients are booted, the clients request the upgrade time. Because of that, we don't need to manage the clients separately.

5.1 Structure

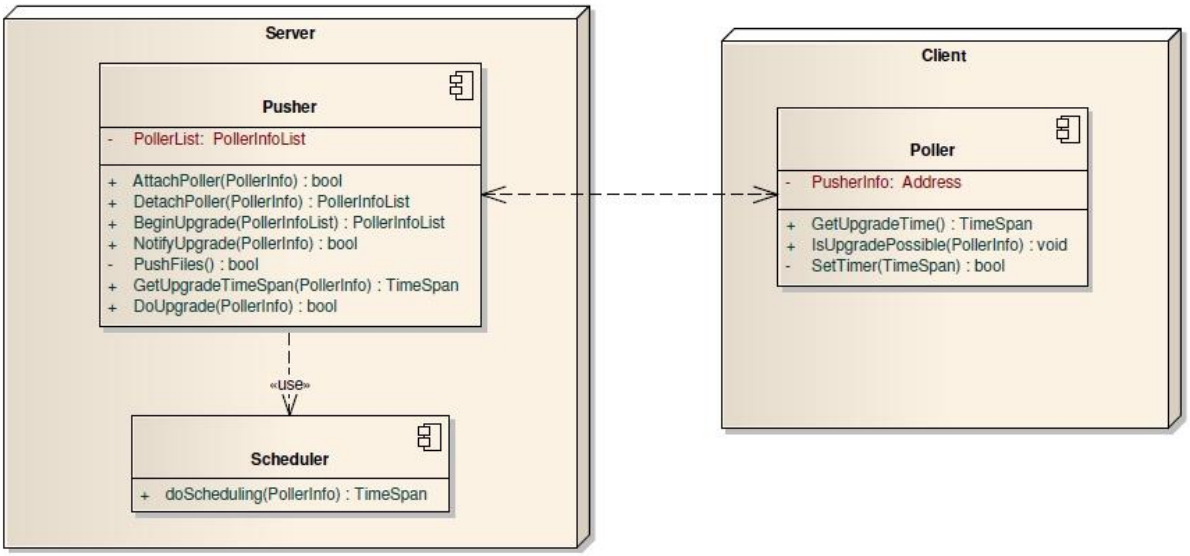


Figure 1. Half-Push/Half-Polling Structure

Pusher component is the advanced component that improves existing push function. This component controls poller list that demanded the upgrade and allocates upgrade time.

- AttachPoller – This function registers new poller(client). It means that the goal of this function is to add poller to the PollerList as scheduling target list.
- DetachPoller – this function is for deleting poller from PollerList.
- BeginUpgrade – Entrypoint to start the upgrade. And it is exposed interface to the external source which requested the upgrade.
- NotifyUpgrade – pusher notifies the upgrade process to selected pollers.
- GetUpgradeTimeSpan – poller calls this function to assign the upgrade time from pusher after receive the upgrade request through NotifyUpgrade from pusher.
- DoUpgrade – to call PushFiles() for upgrade when assigned upgrade time of poller is 0.
- PushFiles – to forward the actual upgrade list of files to poller.

Poller is component which needs the upgrade regularly. It has to have a pusher address that poller can access all the time.

- GetUpgradeTime – Receive the upgrade time from pusher.
- SetTimer – Save the time allotted from pusher. However, it does not run the upgrade directly at the assigned time. Instead of that, poller calls pusher's GetUpgradeTimeSpan() function in case of server's throughput would reach the limit.

The purpose of scheduler component is to conduct a scheduling strategy for distributing the upgrade time. The scheduler component can adopt various scheduling strategies [1][12] according to client(poller).

5.2 Dynamics

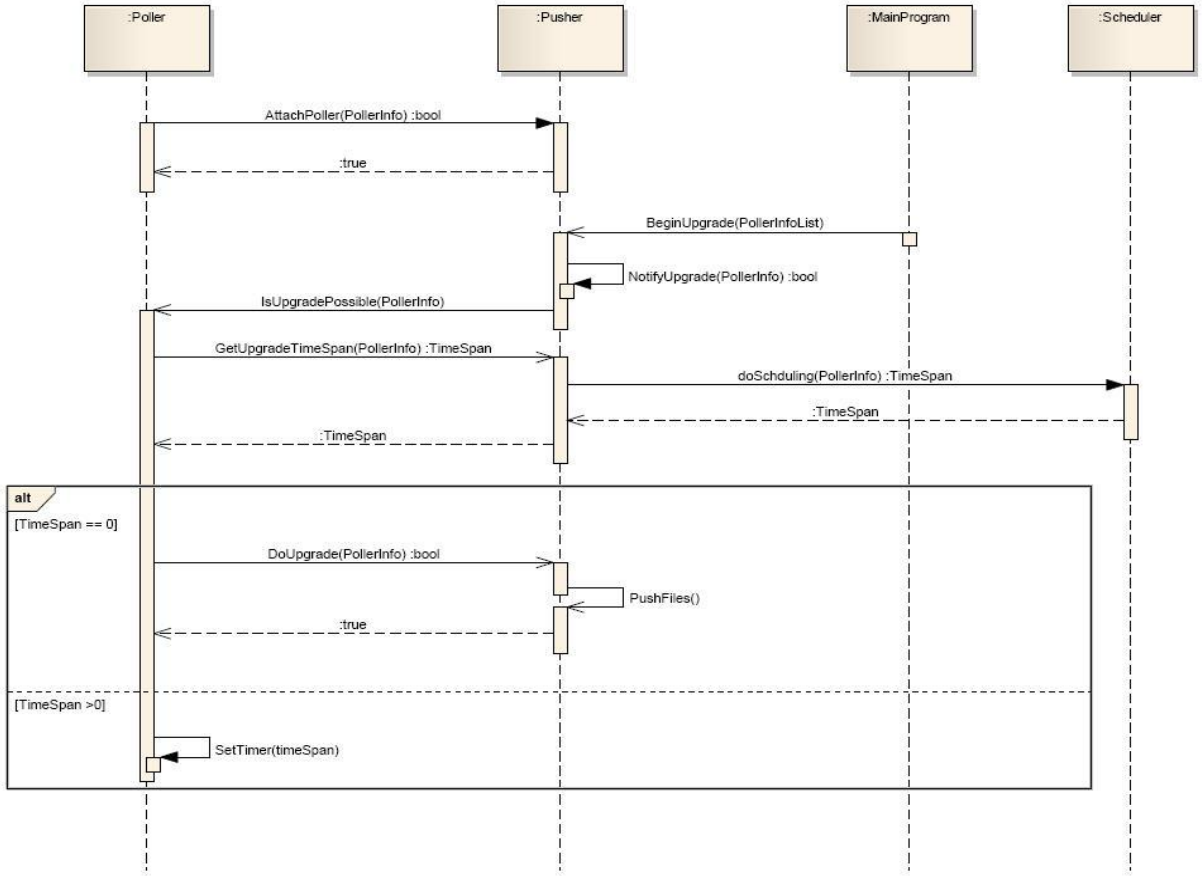


Figure 2. Half-Push/Half-Polling Upgrade Flow Chart

As Publisher-Subscriber [7], Poller(client) can register and cancel for executing the upgrade service by passing the it own reference to pusher(server).

The BeginUpgrade() function is executed by The external program to pusher.

Then, pusher sends the upgrade information (version etc.) through IsUpgradePossible() function to pollers. Next, poller calls the GetUpgradeTime() function when it needs the upgrade after compare poller's version with pusher's version. After that, the scheduler is called for allocating the upgrade time interval internally. If the upgrade time as the TimeSpan is the specified time such as 3:04 Pm, the complex time synchronize mechanism [9][10] is needed between pusher and poller or pollers. To avoid that, we apply the time interval as the TimeSpan like 300ms, 400ms.

When the assigned upgrade time is 0, poller calls the DoUpgrade() function. But if the assigned upgrade time is more than 0, poller waits until the time is 0. It does not run the upgrade directly at the assigned time. Instead of that, poller calls pusher's GetUpgradeTimeSpan() function in case of server's throughput would reach the limit. So poller gets the new time for server availability.

6. Implementation

Step 1 : grasp the characteristics of upgrade targets(pollers).

It is possible that clients (pollers) that always connect to the network can be selected as the upgrade target. However, we should adopt the polling method for clients who often disconnect to the network such as P2P.

Step 2 : choice the scheduling algorithm considering Quality of Service (QoS).

The following QoS[13] should be regarded at the upgrade time.

- Flexibility; any part of system can be upgraded.
- Robustness; the risk of error and crash should be minimized.
- Ease of use; upgrade process should be concise.
- Low overhead; It should minimize the impact on system performance.
- Cost; it should minimize the cost of upgrading.
- Independence; modules which are not related to upgrade should not be given any affect.
- Reliability; Robustness is related to risk during upgrade, Reliability is related at the end of the upgrade. In other words, it should be able to trust that upgrade is done correctly.
- Integrity; it should maintain one status of either complete upgrade or nothing. To upgrade only part of files should not be allowed. This means rollback function is supported when upgrade error invokes.
- Continuity: the upgrade should be run without interrupt.

It is important to get a handhold on QoS for the upgrade. The right scheduling strategy is determined according to the system. Such as the system deals with a variety of upgrade version or deadline is important for the scheduling in a system like Realtime system.

Step 3 : Decide the message exchange format.

If a standard message format is used, such as XML for interoperability, a conversion process is required like Marshaling/Unmarshaling. It is not suitable for a system in which quick response is needed because of limited resource. In that case, despite system dependency, we should require message transfer format based on protocol using Binary Method Table [7] for performance guarantee such as COM+, OLE.

Step 4 : Should build care of information from Poller to Pusher for extension.

The information which is sent to pusher can be changed according to scheduling algorithm or poller's feature. Various pollers are added in the system and thus, the exchange information is designed for extension. For that reason, it should consider Composite Message[4] or Parameter Object pattern[5].

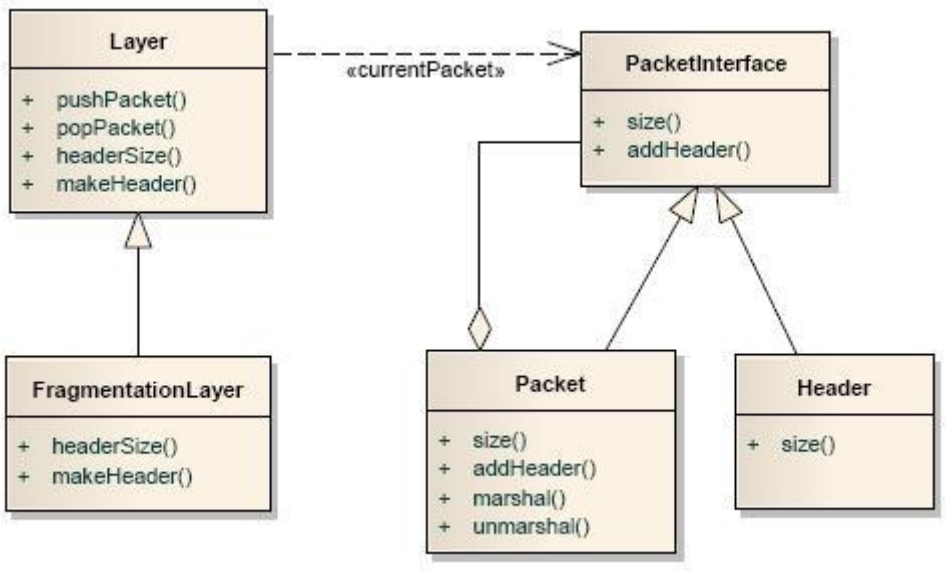


Figure 3. Composite Message Pattern

In Composite Message pattern, the exchange message can be added through Pipe or Filter. And then, it exchanges or extends protocol easily. So it is not suitable for Embedded System which has limited resource sand requires quick response.

Step 5 : Consider the upgrade to type, group, or kind of dependencies between the pollers.

Most case, the system has various pollers(For example, the company has many different kinds of mp3 devices or the company manage various version of electronic products). In addition, we should consider dependency among pollers for certain service. In this case, to solve the problem, we can make various groups using Event Channel [1] between pusher and poller as figure 4.

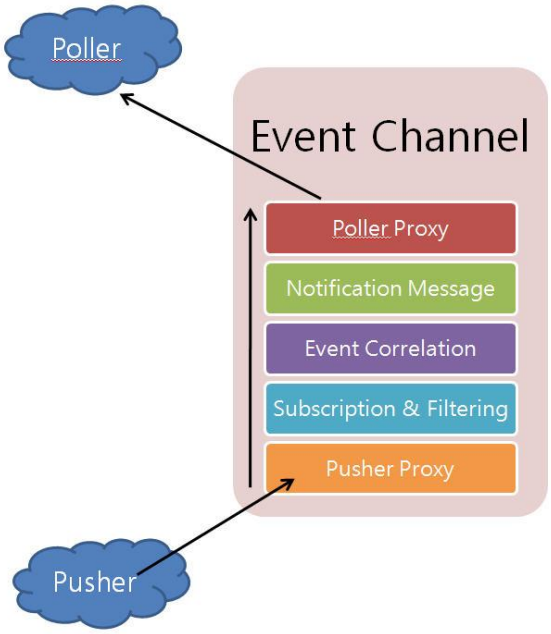


Figure 4. The Instance of Event Channel

```
class EventChannel
{
……..
// Managed poller list
   private List ConsumerList;
   private List FilterList;

   public bool AttachConsumer();
   public bool DetachConsumer();

//assign the filter for grouping of various pollers(upgrade target)
   public bool AttachFilter();
   public bool DetachFilter();

//Start the EventChannel service.
   public bool Run();

//send the poller's information ( i.e. device own ID, state information, App version, Framework
   version information etc.)
   public void NotifyUpgrade();
   };
```

Above the picture and source are the examples of Event Chanel structure.

The direct dependency between pusher and poller is removed because various pollers are managed with Event Channel as above. In other words, more flexible structure has occurred to change when add/ remove the new poller or add/change/remove the target upgrade group. In addition, using the filters the upgrade targets(pollers) can be grouped into various forms as follow items.

- In case of appointing type of upgrade poller. For example, the type is 3536 and 3836 – "NodeType:3536", "NodeType:3836"
- In case of appointing resident such as PLoP Apartment Number 101 – "AptNum:0101"
- In the event of appointing upgrade poller group. For example, the range of group is between PLoP Apartment Number 101 and PLoP Apartment Number 112 – "Range:AptNum:AptNum:0101:AptNum:0112"
- In case of appointing type for permission access to Apartment. For example, the type is 3536 as entrance system at PLoP Apartment Number 101 – "RangeAptNum:AptNum:0101:NodeType:3536"

Step 6 : Use the Timer or WatchDog manage the time information from pusher.

If pusher and poller have absolute time as TimeSpan, the complex time synchronize mechanism is needed. However, the time interval is used as TimeSpan in the Half-Push/Half-Polling pattern. Instead of that, either Timer or Watcher (WatchDog)[2] is needed to control the time interval. In addition, current time and the time interval (TimeSpan) are necessary to store on a file or DB in case of system failure. So, we can know whether the upgrade request is needed with the information when the system restarts. If the current time is greater than the time that we stored on the file or DB and then, poller rechecks that the upgrade is possible to pusher.

Step 7 : consider appropriate File Transmission Mechanism for situation.

All systems have to use appropriate mechanism depending on domain or situation. The optimal solution for every situation (Silver Bullet) does not exist. When you need to upgrade/patch many different kinds of devices, it is necessary to consider a variety of network environments. Maintain session, either the pollers which request mass file transfer may exist or the pollers which periodically request small amounts of data may exist. To consider this situation, the File Transmission Strategy is chosen according to type of poller. To resolve these problems, "JAWS: A Framework for High-Performance Web Servers"[3] provides the good solutions. In that study, the asynchronous transfer mechanism (Proactor) like IOCP has bad performance for transferring small file.

7. Variants

When the server (pusher) distributes the upgrade time though scheduler, some of clients are often off-line. Then the server waits the response from clients for a certain time. Because the client is off-line, wrong scheduling is made and the increase of total upgrade time occurs.

To solve this problem, Alive Check Manager that distinguishes whether the target pollers (clients) are alive or not is needed a separate upgrade module.

Alive Check Manager sends cycle to the pollers which are registered. Then those pollers send the Alive message to server according to the assigned cycle. For example, if the alive message does not come over the cycle * N time, it is necessary to change the state of poller the dead and the poller is removed on the upgrade target list.

8. Known Use

- OMG CORBA Event Service

The Event service of RealTime CORBA it is not for upgrade, but the Event Channel method that replaces push method with pull (polling) method as data transmission method.

- Hybrid Push/Pull Download Model in Software Defined Radios.

A Software-Defined Radio (SDR) system is a radio communication system where components that have typically been implemented in hardware (e.g. mixers, filters, amplifiers, modulators/demodulators, detectors. etc.) are instead implemented using software on a personal computer or other embedded computing devices. When a new way of service starts, it is natural to replace existing terminal with new one. However, the SDR performs on the most Software instead of the existing semiconductor. Hybrid Push and Pull [11] which is one of the down models SDR offers is a good example of Half-Push/Half-Polling pattern.

- Samsung Homevita

Homevita, Home networking system at Samsung electronics, adopted the upgrade methods with a mixture of push and polling method. In the Homevita 1.0 version, it took the polling method for upgrade in the Homevita 1.0 version. However, it invoked the overhead by request of many devices simultaneously. To avoid the problem, the system adopted the push upgrade method in the Homevita 2.0 version. After that, overhead was reduced. But the system needed a way to monitor dead devices. In the latest version, a mixture of push and polling method are adopted. It reduced server overhead. And the devices which are alive keep the newest version.

9. Consequence

The advantages of this pattern include:

- Solving the problem of heavy loads in very short periods of time caused by the Polling-based upgrade method and benefiting from the Push method which only upgrades specific clients.
- Reducing network traffic and load of the server/client because it doesn't need to check the version of the server periodically.
- Being able to upgrade specific selected clients by grouping them.

Possible disadvantages are:

- It is not suitable for systems like vaccines that require all devices to be upgraded in case of emergency because the scheduling method used takes server load into account.
- It is impossible to upgrade clients if a problem occurs on the server (Pusher) while the program is running. Reliability must be guaranteed by copying Pusher components or using various Fault Tolerance methods[2].
- It is difficult to apply to a system such as P2P, where server and client information changes frequently.

10. Related Patterns

Publisher-Subscriber

Also called the Observer pattern, it is used to synchronize the information between two components-Publisher and Subscriber. Copying the database from Publisher to Subscriber can be a typical example. It is used when the pattern encounters a non-stop talker object, in other words, when overload occurs because of non-stop Polling.

Composite Message

This pattern is used for marshaling/un-marshaling data, extending and adding messages you want to transfer while passing through layers. It is also used to create a transmission protocol for each device (Poller) in environments heterogeneous to the Half-Push/Half-Polling pattern. It is used in various distributed middleware.

Pipe & Filter

This pattern is used when adding or filtering messages you want to transmit flexibly according to the circumstance, used internally in the aforementioned Composite Message. It is also used to filter unwanted data when building an Event Channel.

Broker

This pattern removes direct dependency (location information, platform restrictions, etc.) between server and client. By delegating the location information of Pusher, which is in Poller, to Broker, Poller can remove itself of its direct dependency on the Pusher.

References

[1] Timothy H.Harrison, david L. Levine, and Douglas C. Schmidt, "The Design and Performance of a Real-time CORBA Event Service," Proceedings of OOPSLA '97, Atlanta, Georgia, October, 1997

[2] Robert S. Hanmer, "WatchDog", Patterns for Fault Tolerant Software, WILEY, 2007

[3] James C. Hu, Douglas Schmidt, "JAWS: A Framework for High-Performance Web Servers", Domain-Specific Application Frameworks: Frameworks Experience By Industry, John Wiley & Sons, October, 1999.

[4] Aamond Sane, Roy Campbell, "Composite Messages: A Structural Pattern for Communication between Components", OOPSLA' 95 Workshop on Design Patterns for Concurrent, Distributed, and Parallel Object-Oriented Systems, 1995.

[5] Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts, "Refactoring : Improving the Design of Existing Code", Addison-Wesley Professional , 1999

[6] Douglas C. Schmidt, Michael Stal, Hans Rohert, and Frank Buschmann, "Proactor", Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, WILEY, 2000.

[7] Frank Buschmann , Regine Meunier , Hans Rohnert , Peter Sommerlad , Michael Stal, "Broker", Pattern-Oriented Software Architecture Volume 1: A System of Patterns, WILEY, 1996

[9] F. Cristian, "Probabilistic Clock Synchronization, Distributed Computing, vol. 3.

[10] R. Gusella, S. Zatti: "The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3 BSD", IEEE Transactions on Software Engineering, Vol. 15, July 1989

[11] Jamadagni, Satish., Umesh M.N**., "**A PUSH download architecture for software defined radios**",** 2000 IEEE international symposium on personal wireless communication

[12] Sameer Ajmani, Barbara Liskov, Liuba Shrira, "Scheduling and Simulation: How to Upgrade Distributed Systems"

[13] Michael Hicks, Jonathan T. Moore, Scott Nettles, "Dynamic Software Updating", ACM Transactions on Programming Languages and Systems (TOPLAS) ,Volume 27 , Issue 6