Deployment Pattern

YOUNGSU SON, Samsung Electronics
JIWON KIM, Samsung Electronics
DONGUK KIM, Samsung Electronics
JINHO JANG, Samsung Electronics

Software that is deployed into the market needs to be continuously evolved. In order to be long-running, it is necessary to incorporate new client requirements throughout the lifetime of the product. Professional Software Deployment is essential for software evolution, and many vendors therefore provide deployment applications.

Categories and Subject Descriptors: **D.2.11 [Software Engineering]**: Software Architectures—*Patterns*; **D.2.7 [Software Engineering]**: Distribution, Maintenance, and Enhancement—*Enhancement, Extensibility;* **H.3.4 [Information Storage And Retrieval]**: Systems and Software—*Distributed systems;* **K.6.3 [Management of Computing and Information Systems]**: Software Management—*Software maintenance;*

General Terms: Design

Additional Key Words and Phrases: Deployment, Update, Upgrade, Downgrade, Push, Polling, Half-Push/Half-Polling

ACM Reference Format:

Son, Y.S., Kim, J.W., Kim, D.G. and Jang, J.H. 2011. Deployment Pattern. in 18th Conference on Pattern Languages of Programs (PLoP), (Portland, Oregon, USA, 2011), Hillside Group, 9 pages.

1. INTRODUCTION

Software must reflect the changes of the real world. If not, software will decay faster over time. No matter how well-made, software must always be revised based on the client's requirements changes and the needs for new services. Due to this, many applications currently support deployment services in order to improve the customer satisfaction. With this deployment service, the clients can use the latest services without installing a new program every time. In this paper, we discuss an important pattern for deployment services which is applicable to various systems that consist of various types of devices. Here we see that fixes, partial updates, upgrades, and installing new programs are almost the same for providing any new service.

2. CONTEXT

Let's assume that we are developing an office automation system for buildings located closely together in a downtown area. There are various types of devices in the system and they are connected to a wired or wireless network. In addition, the requirement is to keep the software in each device up-to-date. The server will provide the latest software via a client-server model.

Author's address: Y.S. Son, Samsung Electronics Co. Ltd., Suwon, Republic of Korea; email: indigoguru@gmail.com; J.W. Kim, Samsung Electronics Co. Ltd. KiHeung, Republic of Korea; email: yesdiz@gmail.com

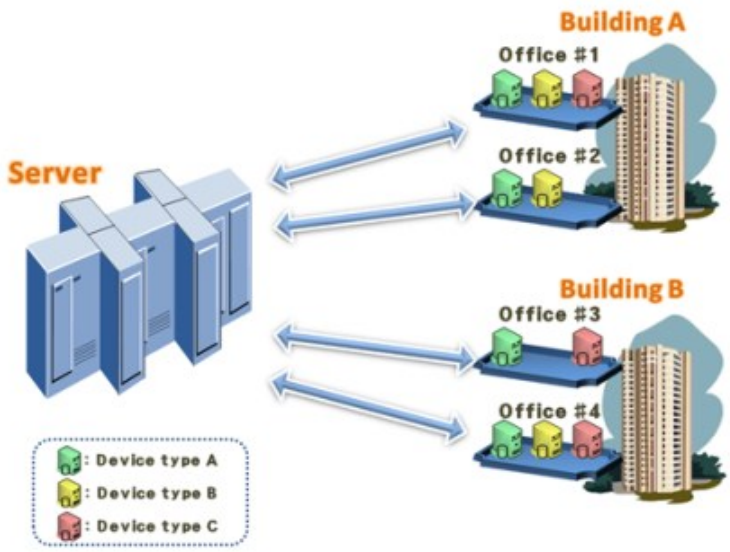

Fig. 1. Building system which consists of various types of devices

Some devices should be updated emergently. For example, the when Security system's server or module is updated, emergent update is quite needed. However, printer driver update in office is not urgent.

2.1 Conventional Approach: Push and Pull Concepts

In the Client-Server Model, Push and Polling are general methods to notify the updates at the server to the clients. Push refers to actively updating clients by the server. The server requests access to the clients and pushes data to them. The advantage of this approach is that you can fully utilize the server resources within the limit of network bandwidth. However, this method assumes that the client will always be alive. And it also requires the server to keep information about clients. Figure 1 shows how the Push method works. When some event (ex; upgrade) occurs and the server needs to connect to the clients, the server can control the workload autonomously in consideration of its capacity and the bandwidth. Numbers in the picture represent a sequence of operations.

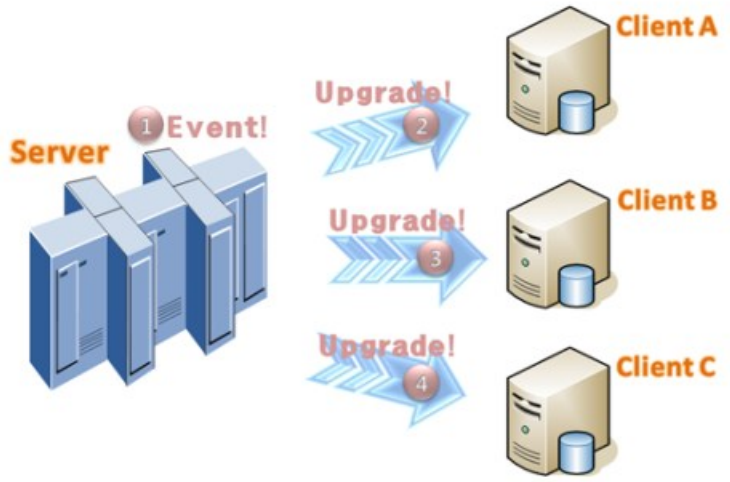

Fig. 2. Push Upgrade Method

On the other hand, when using polling method, a client has the initiative. The client requests data transmission from the server. Because the server is more stable and using this method guarantees the server is alive, the chance of problems is very low. However, in the Polling method, as opposed to Push, the client checks repeatedly whether any event has occurred or not. In the worst case, all clients may attempt to access the server at the same time because there is no fixed order. The figure below shows such case.

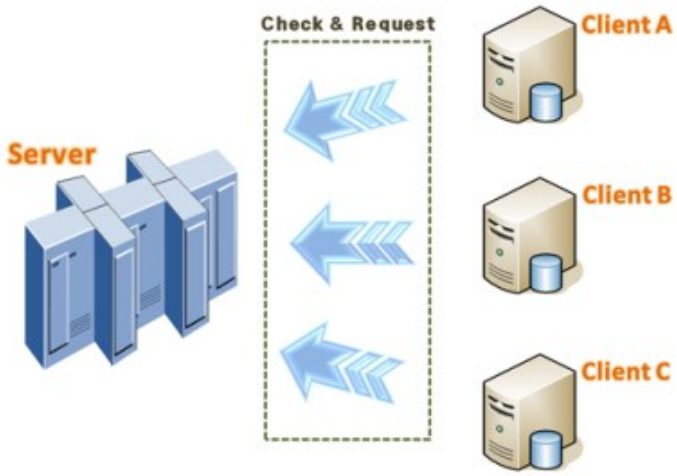

Fig. 3. Poll Upgrade Method

3. PROBLEM

How do you deploy software updates from a server to a client base that consist of multiple devices running a combination of applications and that are distributed on several locations?

4. FORCES

When software in use is distributed, there are many considerations.

- A case that each client group uses different program version.
- A case that each client group needs to be upgraded in different timing.
- A case that each client group needs to use different distribution strategies.
  (Ex. Every 3 hour, every week)

So, just compositing of Push and Polling method is insufficient to solve these forces.

Let's think of management system for businesses. They can have different service version and distribution strategies according to each client group.

Furthermore, some of companies might want to upgrade in not busy time, and not want to download the whole program every time new version released but only download changed module to avoid server's over load. Clients should pay much money for download not impossibly.

5. SOLUTION

The main parts of the solution are the Ticket and the TicketSolver. Ticket describes information which is needed for the deployment operation like the file/module/component to deploy, deployment strategies (broadcast, staged, future) and the channel type. The TicketSolver uses Ticket information to support various deployment ways by binding a channel with strategies. Also TicketSolver makes a lot of channels to meet the requirement of each specific client.

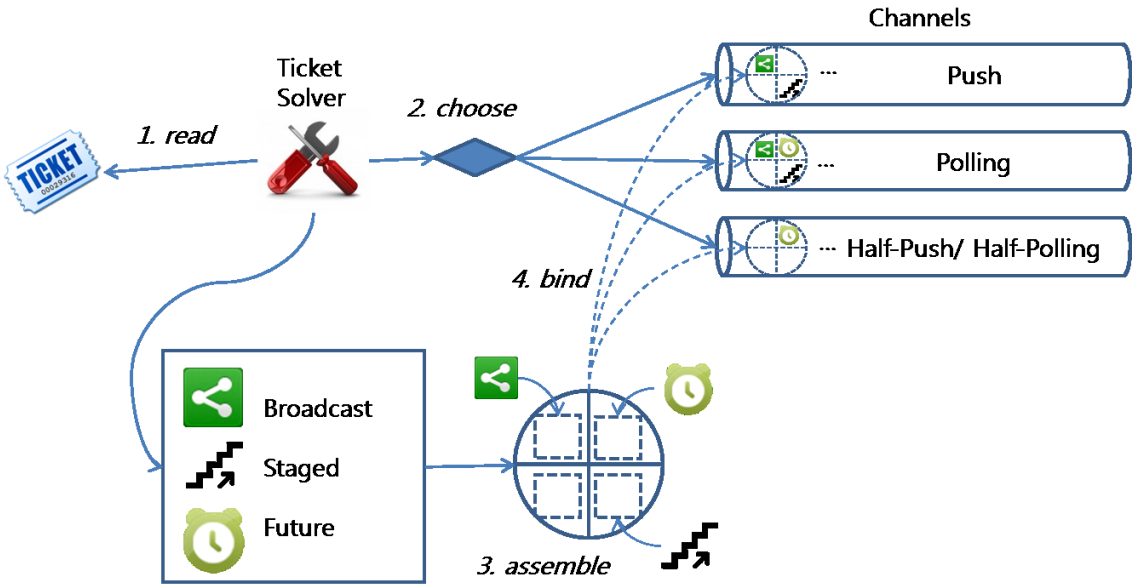

Fig. 4. Deployment Pattern Map

For example, when choosing channel type, we can think this.

- Push Channel: By Service Provider, emergent and critical update like hotfixes.
- Polling Channel: By Users, may need agreement from users, may be opt-in updates.
- Half-Push/ Half-Polling (Son et al. 2009) Channel : Regarding update service stability by postponing several update request with no time constraint. Like a Windows Update, the server can update clients easily within the limits of its resources regardless of the state of clients.

6. STRUCTURE

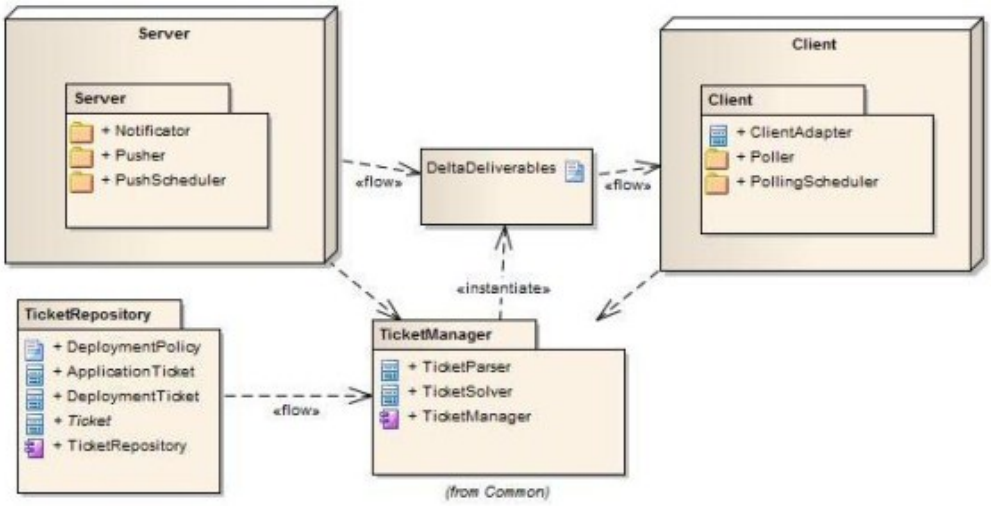

Fig. 5. Overview Structure of the Deployment Pattern

6.1 Ticket

The upgrade Ticket can be used for software distribution with various distribution strategies according to each client group. It prevents server upgrade overload by managing files and user setting information, and provides optimized and reduced upgrades based on the system environment. The upgrade Ticket consists mainly of an Application Ticket and a Deployment Ticket.

- Application Ticket

The Application Ticket includes information about the Software to be distributed, such as dependency of application, and assembly, file, and permission. The ticket manages file information which ensures that only the changed module will be upgraded and downloaded from the distribution server.

- Deployment Ticket

The Deployment Ticket includes a distribution strategy : “Broadcasting, Staged, and Future” and a type of update “Push - Publisher/Subscriber” (Buschmann et al. 1996), “Poll”, “Half-Push/Half-Poll” (Jamadagni and mesh 2000), (Son et al. 2009).

6.2 Ticket Solver

According to the Deployment strategies selected (Push, Poll, or Half-Push/Half-Poll), the Ticket Solver dynamically deploys and redeploys correspondent components in runtime. It is based on Composite Message (Sane and Campbell 1995) and Pipe&Filter (Buschmann et al. 1996).

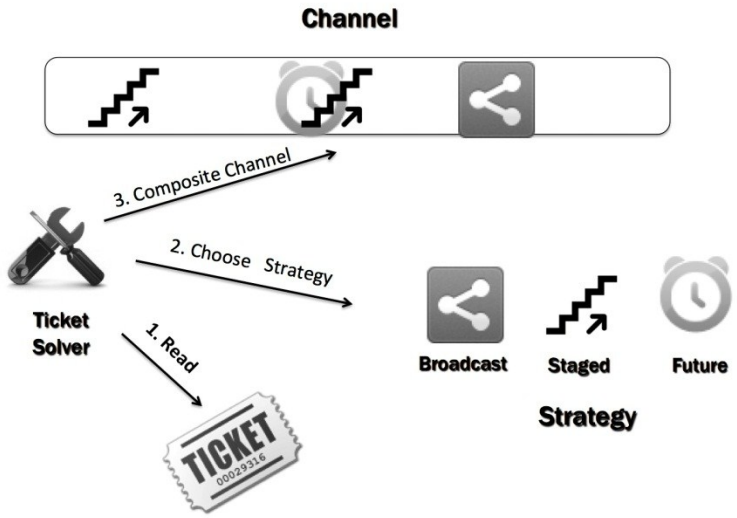

Fig. 6. Overview of channel binding

For example, when assembling strategy, we can think this.

- Broadcasting(Microsoft 2005) : A new version is placed on the deployment server and everyone starts downloading it in the same time frame immediately after it is published.
- Staged(Microsoft 2005) : A new version is placed on the deployment server, but different groups are expected to download the update at different times.
- Future(Microsoft 2005) : A new version is available, but it needs to be applied at a specific point in the future.

6.3 Scheduler

There are two schedulers. Half-Push/ Half-Polling uses both schedulers.

- Push Scheduler

The Push Scheduler is used for simultaneous update of all clients. (It is also used in the Half-Push/Half-Poll solution explained later in which case it only forwards scheduling information). Clients who want to get updated register their own version information and access information in the Push Scheduler. The connection is 1:1 and all the clients are registered in a list (Publisher-Subscriber) (Buschmann et al. 1996) and the Push Scheduler deliver modules to clients.

- Polling Scheduler

All clients who need to simultaneously ensure the consistency, as well as systems that won't upgrade use the Polling Scheduler to download available module updates at the time of request.

6.4 Notificator

The Notificator is used with the Push Update. In case of a Push Update, it passes a Channel made by the TicketSolver. The Channel is used for comparison of version information and extraction of modules which will be updated. In other words, DeltaDelieverables are derived. These modules are used to notify to Polling Scheduler.

7. DYNAMICS

7.1 Dynamic Channel Composition by Ticket information

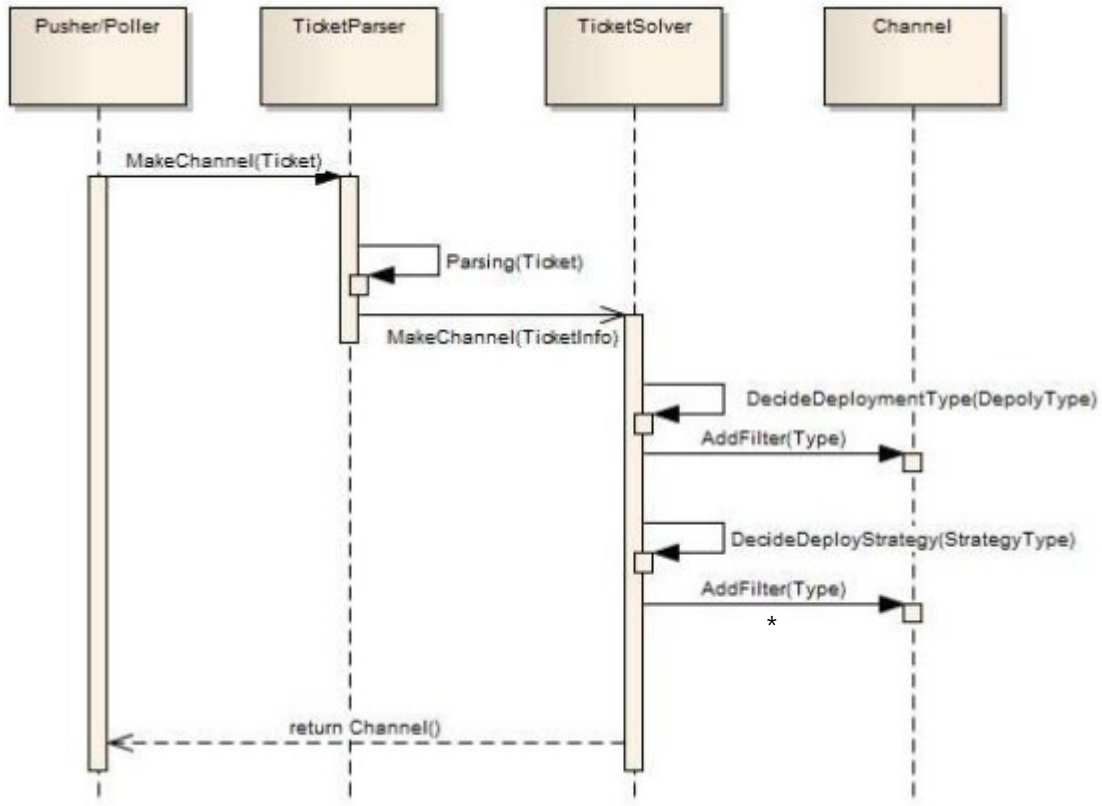

Fig. 7. Dynamic Channel Composition

1. When a Ticket is added to TicketRepository, the ticket information is delivered to a correspondent object based on Deployment type.
2. The TicketParser analyses Ticket information about how to distribute (push, poll, half-push/half-poll), and what strategy to use (broadcasting, stage, future).

3. This information is delivered to the Solver (Distributer) and then the Filter is made according to Distribution method and Strategy. As per a strategy decided such as Push or Poll, a channel is created by dynamically composed filters.

7.2 Push Deployment

1. Clients to be updated register themselves to Push Scheduler.
2. A connection is managed as 1:1 and client list is registered in server.
3. Each client list is registered in server, and the Push Scheduler is in charge of distribution of one client.
4. A module's information for distribution is described in Application Ticket and Push for one client is done as per this information.

7.3 Poll Deployment

1. The Poll Scheduler sends the Application Ticket information in an upgrade request program or it gets a ticket from Ticket Manager in remote.
2. The Poll Scheduler dynamically makes a channel as per policy after parsing the Ticket information by Ticket Parser.
3. The client downloads modules from the Server.

7.4 Half-Push/Half-Polling Deployment

1. The Ticket is delivered from TicketManager to TicketSolver, and then the channel is dynamically created by TicketSolver.
2. Push Scheduler has information of Pollers to upgrade and it asks to Poller if currently client is upgradable.
3. Push Scheduler extracts the Poller Lists from ticket describing Deployment policy.
For example, Broadcast should update all Pollers and Future should only delegate clients which reached reserved time to scheduler.
4. Push Scheduler sends Timespan about when to upgrade to current Pollers.
5. When the reserved time comes, Pollers will request the upgrade to the Pusher.
6. Pusher checks the server's throughput.
    - If Server is still busy, resend extended Timespan to Poller.
    - If Service is available, Pusher releases DeltaDeliverable to Poller.

8. KNOWN USE AND VARIATIONS

KNOWN USES

- ZeroInstall (Leonard 2005)
ZeroInstall is a decentralized cross-distribution software installation system. Other features include full support for shared libraries, sharing between users, and integration with native platform package managers. ZeroInstall provide Solver. Solver creates architecture in runtime as per Deployment Policy. Also it offers Application and Deployment Ticket as Feed and Policy objects mentioned in this paper.

- Clickonce (Microsoft 2005)
The core principle of ClickOnce is to bring the ease of deployment of web applications to the Windows user. In addition, ClickOnce aims to solve three other problems with conventional deployment models: the difficulty in updating a deployed application, the impact of an application to the user's computer, and the need for administrator permissions to install applications.

A ClickOnce deployment is controlled through the use of two XMLmanifestfiles: a deployment manifest and an application manifest. The manifests are in the same XML format as the Side-by-SideAssembly implementation. The deployment manifest (*.application file) describes the deployment model: the current

version, update behavior, publisher identity along with digital signature; this manifest is intended to be authored by administrators who handle deployment.

The application manifest(*.exe.manifest file) describes the application assemblies, dependent libraries and lists permissions required by the application. This file is intended to be authored by the application developer .In order to launch a ClickOnce application, a user clicks on its deployment manifest file.

- FF Clickonce (Dobson 2006)
This, a Clickonce .NETonvariant, is created to distribute the latest version of Firefox extension by Firefox Community

VARIANT

-OMG CORBA Event Service (Harrison et al. 1997)
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. In order for various message deliver, this paper suggest not only Push, Pull(Polling) but a variety of supplier and consumer such as passive pull supplier - active pull consumer.

9. CONSEQUENCES

The advantages of this pattern include:

- Dynamic deployment components and supports various upgrade methods.
- Being able to upgrade specific selected clients by grouping.

Possible disadvantages are:

- Necessity of maintenance due to a higher complexity than simple update methods.
- Necessity of many distribution components (Client and Ticket Manager as well as Server)
- It is difficult to apply to a system such as P2P, where server and client information changes frequently.

10. RELATED PATTERNS

Publisher-Subscriber (Buschmann et al. 1996)
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 (Sane and Campbell 1995)
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 (Buschmann et al. 1996)
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.

Half-Push/Half-Polling (Son et al. 2009)
Push causes much less load because push can upgrade specific client but there is cumbersome monitoring to keep stopped clients on latest version. The Half-Push/Half-Polling pattern mixes these two different ways, keeping their advantages, eliminating their disadvantages.

ACKNOWLEDGMENTS

We thank our shepherd Kiran Kumar Reddy and Lise Hvatum for his valuable comments that contributed to improve this paper. The EVA (Pattern Evangelist Group), James Chang provided useful improvements and corrections.

REFERENCES

Ajmani, S., Liskov, B. and Shrira, L. 2003, Scheduling and Simulation : How to Upgrade Distributed Systems. In *Ninth Workshop on Hot Topics in Operating Systems (HotOS-IX)*, (Lihue, Hawaii),
Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. 1996. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*, WILEY.
Cristian, F. 1989. Probabilistic Clock Synchronization, *Distributed Computing, vol. 3.*
Dobson, J. 2006. FFClickOnce
http://www.softwarepunk.com/ffclickonce/
Fowler, M., Beck, K., Brant. J., Opdyke, W. and Roberts, D. 1999. *Refactoring : Improving the Design of Existing Code.* Addison-Wesley Professional.
Gusella, R. and Zatti, S. 1989 The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3 BSD. In *IEEE Transactions on Software Engineering, Vol. 15*, (July 1989).
Hanmer, R. S. 2007. WatchDog. *Patterns for Fault Tolerant Software*, WILEY.
Harrison, T. H., Levine, D. L. and Schmidt. D. C. 1997. The Design and Performance of a Real-time CORBA Event Service. In *Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications(O OPSLA'97),* ACM, New York, USA
Hu, J. C., Schmidt, D. C. 1999. JAWS: A Framework for High-Performance Web Servers, *Domain-Specific Application Frameworks: Frameworks Experience By Industry*, John Wiley & Sons, (October 1999).
Jamadagni, S., Umesh M.N. 2000. A PUSH download architecture for software defined radios in *IEEE international Conference on Personal Wireless Communication*.
Leonard, T. 2005. ZeroInstall
http://0install.net/, http://en.wikipedia.org/wiki/Zero_Install
Microsoft. 2005. ClickOnce - Microsoft Deployment Application
http://msdn.microsoft.com/en-us/library/ff650190.aspx, http://en.wikipedia.org/wiki/ClickOnce
Sane, A. and Campbell, R., 1995. Composite Messages: A Structural Pattern for Communication between Components, In *Proceedings of the Workshop on Design Patterns for Concurrent, Distributed, and Parallel Object-Oriented Systems(OOPSLA'95)*. ACM
Schmidt, D. C. Stal, M., Rohnert, H., and Buschmann, F. 2000. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*, WILEY.
Son, Y. S., Ko, S. W., Jang, J. H., Lee, H. J., Jeon, J. M. and Kim, J. S. 2009. Half-Push/Half-Polling. In *Proceedings of the 16th Conference on Pattern Languages of Programs*