A Comparative Study of using Object oriented approach and Aspect oriented approach for the Evolution of Legacy System

Dr S.A.M .Rizvi, Jamia Millia Islamia, New Delhi
Zeba Khanam, PhD Scholar, Jamia Milia Islamia, New Delhi

Abstract

Legacy systems are vital to an organization, and sometimes form the backbone of an organization, yet their maintenance and evolution had been an area of research for a long time. Besides being costly to maintain, legacy systems often lag behind changes in the businesses they support. The challenge in today's environment is to develop a methodology to migrate older systems to newer, more cost effective client-server distributed processing platforms that support standards-based modular architectures. One approach is to employ a "wrapper" of code that surrounds the existing legacy code, turning it into an object. This could be stated as an object oriented approach to legacy systems. However, there are many other paradigms that a legacy system might adopt. Aspect-oriented technology is another emerging programming paradigm that is receiving considerable attention from research and practitioner communities alike. Nowadays much of the work is carried on, on developing different methodologies to enable aspect oriented programming to be applied to legacy systems. In this paper, we try to analyze the impact of object oriented technology and aspect oriented technology on legacy systems and the environment that is required to implement the two paradigms. The advantages and disadvantages of both the paradigms have been explored, and a comparative study of both the paradigms is done and analyzed in the light of legacy systems.

Introduction

Many existing systems are expensive to maintain because their priority mainframe-based technologies are no longer current and do not adequately support their users' processing needs. Downsizing host-based applications to smaller, less expensive systems may provide cost savings on several levels and lead to increased end-user efficiency, due in part to the readily available desktop computing power that may have already been purchased. Even some client-server systems developed in recent years have reached a point where modernization may be necessitated because rapid-paced technological advancements have rendered their hardware obsolete.

Migration of such legacy systems to standards-based open system environments is a formidable challenge. The concept presented here is intended to provide an approach to meet the challenge of legacy migration based on new and exciting technologies [4]. This paper begins with the discussion of both the paradigms, then it emphasizes upon the applicability area of both the paradigms in legacy system .The next section deals with a comparative analysis of both the paradigm in context with their applications to the legacy system.

Object-Oriented Approach to Legacy System Migration

This section presents a discussion of various ways in which object oriented approach can assist in evolution of legacy system. One of the approaches is based on the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) for migration of legacy systems. The OMG is a consortium established to remote industry guidelines and object management specifications in order o provide a common framework for the development of distributed applications. However, as with any evolving technology, there are competing standards. Microsoft's Object Linking and Embedding (OLE), and the pen

Software Foundation (OSF) Distributed Computing Environment DCE) are similar alternative approaches to legacy migration. [6]

Systems built upon the principles of an object-oriented architecture maximize portability, reusability, and interoperability of software, resulting in a true open system solution.

By using the encapsulation or "wrapper" approach, irreplaceable system applications can be transformed into object-oriented components for a modular architecture suitable to a heterogeneous, distributed processing environment. One product which can facilitate the object-oriented approach is the Universal Network Architecture Services product (UNAS) developed by TRW. The fully executable framework generated by UNAS changes the way distributed systems are built. In addition, UNAS serves as middleware, a layer of software that sits between the operating system and the application, in effect hiding the complexities of operating systems, hardware platforms, and network protocols.
The OMG describes an object-oriented architecture as being developed using the following elements: the Object Request Broker (ORB), Common Object Services (COS), Common Facilities, and Application Objects. A noticeable industry trend appears to support implementation of OMG standards as a mechanism to achieve a truly object oriented distributed system. This approach presents an implementation that can be consistently applied to revamping the architectures of legacy systems and can be used as a blueprint for the development of new distributed systems. Using a modular, component-based architecture should also result in reduced software development and maintenance life cycles and related costs.

A major drawback of this approach is that a specific wrapped legacy code may not be reusable in systems of similar functionality because it was not originally created with reuse in mind. The ease with which objects can be generated may result in uncontrollable application growth, unnecessary complexity, and sloppy development.

The next section deals with migrating Legacy Systems to the Web that is one of the main concerns of enterprises looking for more flexible distributed application environments.

Extending UML for the migration of Legacy Systems to the Web

This migration process comprises the construction of a Web Interface that needs to interact in an arbitrary complex manner with pre-existent business logic modules, which must pay off prior investments. These Web Engineering concerns have been already addressed with UML,

Modeling the integration and interference of design of business logic and Web Interface design is the key factor for getting successful Web Applications. Some proposals [14] exist for the definition of interface and integration with logic that are device and technology independent. Also, business logic concerns have already been partially addressed in a number of Advanced Software Production Environments [13] that use Model Based Code Generation techniques, many of them based on UML-compliant [15] models. One more approach known as OO-H (Object-Oriented Hypermedia) Method [9], aims at extending such UML-Compliant environments with two new features: navigation in heterogeneous information spaces and connexion with pre-existent logic modules. Although the aspects such as service composition, asynchronous execution of services, security concerns or very sophisticated front-ends have not been taken into account, still the new capabilities will be added as the number and type of modeled applications increases.

Hence, we have briefed up certain issues related to the migration of legacy system to object oriented environment, their advantages and drawbacks. The next section deals with the impact of Aspect oriented programming on the evolution of legacy systems.

Aspectual Analysis of Legacy Systems

Aspect-oriented programming (**AOP**) is a programming paradigm that increases modularity by allowing the separation of cross-cutting concerns. AOP states that programming languages based on any single abstraction framework, procedures, constraints, whatever -are ultimately inadequate for many complex systems[7] In AOP, the different aspects of a system behavior are each programmed in their most natural form, and

then these separate programs are woven together to produce executable code.

For example, code that implements a particular security policy would have to be distributed across a set of classes and methods that are responsible for enforcing the policy. However, with aspect-oriented technology, the code implementing the security policy could be factored out from all the classes to an aspect [8]

Logging is the archetypal example of a crosscutting concern because a logging strategy necessarily affects every single logged part of the system. Logging thereby *crosscuts* all \logged classes and methods.

AspectJ , that was developed for java has a number of such expressions and encapsulates them in a special class, as aspects. Soon even procedural languages like C and COBOL also started getting their aspect languages like Aspect C, Aspect C++, Aspicere, Weave C, C4, TinyC, etc.

Approach to Dynamic Software Evolution

AOSD also supports dynamic evolution of legacy systems. Peter Ebraert has proposed a solution that allows systems to remain active while they are evolving [10]. He has presented a preliminary reflective framework that allows dynamic evolution of separate concerns. The system evolves in 2 steps. In a first step, the application's cross-cutting concerns should be removed, so that it is well modularized. Aspect mining and static refactoring techniques were used to detect and separate the cross-cutting concerns respectively. In a second step, the well-modularized application should be controlled at the metalevel by a monitor with full reflective capabilities. Such a monitor merged the ideas of EAOP (Event-based Aspect-Oriented Programming) and partial behavioral reflection with the dynamic capabilities of the Smalltalk language.

Impact of AOP+LMP on legacy software

Bram Adams has proposed in his work a mix of aspect-oriented programming (AOP) and logic meta-programming (LMP) to tackle some concerns of/in legacy environments [11]. The work was carried out in the context of the two major languages in legacy environments -C and COBOL. Tracing in C and business rule mining in COBOL was done smoothly, using LMP as a point cut mechanism in AOP. The Y2K-bug is probably the best-known example of problems related to legacy systems. It is important to understand that at the heart of this was not a lack of technology or maturity thereof, but rather the understandable failure to recognize that code written as early as the sixties would still be around some forty years later. The problem statement certainly presents a crosscutting concern: whenever a date is accessed in some way, make sure the year is extended. Knowing which items are dates and which are not requires human expertise. The nice thing about LMP is that we could have used it to encode this.

Comparative Analysis of AOP and OOPs

The impact of both the approaches has been highlighted in the above sections in some of the areas related to the maintenance of legacy systems. Object-oriented technology provides powerful tools, such as encapsulation or multiple inheritances of objects, which enable programmers to construct more functionality with less code than previous methods. More importantly, it can minimize the impact of change by combining data and the functions associated with it into a single package — the object — thus reducing the amount of time and effort necessary to produce an application and also increases reuse of software [2]. The approach developed by OMG was discussed. The basis for the approach is that existing; proven software is retained, thus eliminating the costs associated with new development. Using a modular, component-based architecture should also result in reduced software development and maintenance life cycles and related costs.

An Object-oriented framework has also been developed to increase the availability of integrated applications without fully replicating the application environment such as the application platforms, programs, and data [12]. Some legacy applications are periodically suspended for data backups, end of period processing, system and software upgrades, and/or maintenance. These scheduled suspensions are usually not acceptable for high quality service-oriented applications. An object-oriented cost effective replication technique is used for increasing the availability of networked

application integration during a scheduled unavailability of one or more involved applications [12].

While many standard object-oriented languages do a good job of clearly capturing the behavior of objects, they do a less good job of capturing structural and behavioral invariants, such as object gets a pop message, send this other object a refresh message. Many linguistic mechanisms have been developed to deal with special cases of this problem (i.e. before/after methods), but a great deal of the complexity in real world code still appears to come from cases where the language fails to provide adequate support for a secondary, but still important, aspect of a system.

Aspect Oriented Programming (AOP), an emerging programming paradigm, has been identified as an important technique to aid in re-engineering these systems, because it modularizes crosscutting concerns without actually modifying the original source code ("obliviousness") [1].

Everything that AOP does could also be done without it by just adding more code. AOP just saves writing this code. Assume you have a graphical class with many "set()" methods. After each set method, the data of the graphics changed, thus the graphics changed and thus the graphics need to be updated on screen. Assume to repaint the graphics "Display.update ()" should be called. The classical approach is to solve this by adding *more code.* If there are few set-methods, that is not a problem. But if there are many, then it's getting real painful to add this everywhere. No need to update many methods; no need to make sure to add this code on a new set-method. Only a pointcut is needed.

In addition, refactorings are instrumental for the migration of legacy OO systems to use AOP [5]. Research shows that CCCs represent an important evolution problem in legacy systems, especially if one takes the scale of these systems into account (millions of lines of code). AOP can also be used in the dynamic analysis of the legacy systems that no other paradigm can assist [2].

However, this example also shows one of the big downsides of AOP. AOP is actually doing something that many programmers consider an "Anti-Pattern". The exact pattern is called "Action at a distance". Action at a distance is an anti-pattern (a recognized common error) in which behavior in one part of a program varies wildly based on difficult or impossible to identify operations in another part of the program.

As with all immature technologies, widespread adoption of AOP is hindered by a lack of tool support, and widespread education. Some argue that slowing down is appropriate due to AOP's inherent ability to create unpredictable and widespread errors in a system. Implementation issues of some AOP languages mean that something as simple as renaming a function can lead to an aspect no longer being applied leading to negative side effects.

Conclusion

Analyzing the facts that had been covered in the earlier sections, it can be concluded that AOP does not replace OOP in the maintenance of legacy systems but adds certain decomposition features that address the so-called *tyranny of the dominant composition* (or crosscutting concerns). The ideas and practices of OOP stay relevant. Having a good object design will probably make it easier to extend it with aspects. Although this should always be taken into consideration that the legacy systems should not necessarily include AOP, as it may result in unnecessary code complexity and the programmers might have to face the anti-pattern problem. Therefore AOP should not be seen as a replacement of OOP, but as an approach that makes your code more clean, loosely-coupled and focused on the business logic.

References:

[1] Bram Adams, "Aspect Orientation in the Procedural Context of C", 2006

[2] Bram Adams, Kris De Schutter , Andy Zaidman , Serge Demeyer , Herman Tromp, Wolfgang De Meuter , "Using Aspect Orientation in Legacy Environments for Reverse Engineering using Dynamic Analysis - An Industrial Experience Report",2008

[3] Fatima Beltagui, "Challenges of Aspect-oriented Technology, Features and Aspects: Exploring feature-oriented and aspect-oriented programming interactions", 2003

[4] Gail Cochrane," An Object-Oriented Approach to Legacy System Migration", 1996
[5] Jan Hannemann, "Aspect-Oriented Refactoring: Classification and Challenges", 2006

[6] John Wiley and Sons, "The Common Object Request Broker: Architecture and Specification", Revision 2.0, Object Management Group, 1995

[7] John Irwin, Gregor Kickzales, John Lamping, Jean, Cristina Videiralopes, Chris Maeda"Aspect Oriented Programming", 2000

[8] James M. Bieman, Roger T. Alexander," Challenges of Aspect-oriented Technology, 2004

[9] Jaime Gómez, Cristina Cachero, and Antonio Párraga, "Extending UML for the migration of Legacy Systems to the Web", Spain, 2002

[10] Peter Ebraert and Tom Tourwe, "A Reflective Approach to Dynamic Software Evolution", 2004

[11] Kris De Schutter, Bram Adams, "Face-off: AOP+LMP vs. legacy software", 2007

[12] Nader Mohamed and Jameela Al-Jaroodi and, "An Object-Oriented Approach for High Availability of Applications Integration", United Arab Emirates University, 2007

[13] R. Bell, "*Code Generation from Object Models*", Embedded Systems Programming", 1998

.