A Comparative Study of using Object Oriented Approach and Aspect Oriented Approach for the Evolution of Legacy System

S.A.M Rizvi
Zeba Khanam,
Jamia Millia Islamia,
New Delhi
India

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 keep evolving the older systems so that they are compatible with the real world technological environment. The most common approach was to migrate the legacy code to object oriented code. 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 begin by highlighting the work done in evolving a legacy system using the object oriented approach, then we 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.

General Terms

Evolution techniques of legacy software, comparison, aspect oriented programming, object oriented approach.

Keywords

Legacy software evolution, object oriented approach, aspect oriented approach.

1. INTRODUCTION

As stated in Lehman's first law of software evolution, (Lehman 1974), it is now generally accepted that *E*-type software must be continually adapted and changed if it is to remain satisfactory in use. There are many approaches that an organization may choose to evolve software-intensive systems, depending upon the system, and the technology. During the late 90s, the explosion of web , emergence of net centric computing, significant progress in object technology and system understanding it became easier and economically feasible to evolve the legacy systems to a more flexible and maintainable state. In particular, interface technology, wrapping technology, and network technology were used commonly for the upgradation of existing software assets instead of scrapping them and starting over [17].During this period, most popular approach "software migration" was not often considered feasible. There were significant changes in the approaches that were used to evolve the legacy system due to the emergence of distributed object technology, middleware and wrapping technology. We have elaborated, a few approaches in the next section.

2. OBJECT ORIENTED APROACH TO LEGACY SYSTEM MIGRATION

This section presents a discussion of various ways in which object oriented approach was employed for the evolution of legacy system. One of the approaches was 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. Some other approaches were 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][16].

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

The approach [16] basically targeted mainframe based legacy systems but client server systems were also migrated to object oriented environment using this approach. It worked by using the encapsulation or "wrapper" approach. System applications were transformed into object-oriented components for a modular architecture suitable to a heterogeneous, distributed processing environment.UNAS product, CORBA compliant was used to develop the backbone of distributed system architecture. It facilitated the development of OMG standard distributed environment, however there were limitations to the approach as they are based on evolving standards and the products compliant with these standards may or may not be available, also the legacy systems wrapped that way were not reusable.

The other contribution in this area [20] also aimed at migration of procedurally oriented legacy systems to object oriented platform. The approach used reverse engineering activities to abstract an object-oriented model from legacy code. The methodology was to migrate it incrementally by decomposing existing systems into notable sets of components, each of which potentially implements an object. Each object was re-implemented independently using new object-oriented technologies whereas the old components were used in their original form until the new equivalent objects guaranteed acceptance.

Related work is done by Wong and Li in the same area in which they presented a computer-aided semi-automatic method that abstracts OO designs from the original procedural source code. The results indicated that efficient abstraction of OO design can be obtained out of the original C code [21].

One of the main objectives of maintaining the legacy systems is to migrate the standalone systems to distributed environment. But now many legacy systems are object oriented and even they are not suitable for distributed environment, so work is being done to evolve these systems so that they fit in the new environment. Therefore ,one approach was [19] to migrate the object oriented legacy systems to a component based system as a component-based technology have proven to be more suitable for the new environments due to their granularity and reusability.

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.

3. MIGRATION OF LEGACY SYSTEMS TO THE WEB AND SERVICE ORIENTED ARCHITECTURE

With the tremendous growth of web and the internet and web enabled client server architecture, the traditional information system scenario also changed. The information system working nowadays is no longer working as standalone but are actually working in web based or distributed environment. Therefore, to be compatible with the new environment most of the traditional legacy software system is also migrating to the web enabled client server architecture. The work done by Canfora etal explored the migration of legacy system towards service oriented computing. They used the wrapping methodology to make the interactive functionalities of legacy systems web enabled [23].Another 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. The multi-tier architecture given by Zou [24] provided numerous advantages for legacy system migration and integration with other applications. The technique used specified the component configuration in XML, to provide rich information for the consequent automatic wrapping, integration and searching for the services provided by the legacy component.

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 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 models A case study has been conducted [14] to evaluate the use of MELIS (Migration Environment for Legacy Information Systems) for the migration of legacy COBOL programs to the web. Due to huge demand by the customers, several COBOL systems were required to be migrated to the web. MELIS (migration environment for legacy information systems), was also developed to support the migration process. MELIS has been developed as an eclipse plug-in within a technology transfer project. The results showed that the use of MELIS increased productivity and also reduces the gap between an expert programmer and a novice developer.

Legacy system modernizing using service oriented architecture and web services need to have a user interface that can interact with the SOA/web environment. This problem was solved by developing a wrapper that could interact with the system on behalf of the user [15]. The wrapper behaviour was defined in the form of Finite State Machines retrievable by black-box reverse engineering of the human-computer interface. One more approach known as OO-H (Object-Oriented Hypermedia) Method [9], aims at extending the 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.

Other approaches that have gained popularity in recent years is the modernization of legacy software for service oriented architecture (SOA). One of the major difficulties with the legacy systems is making them interface with new, open and modern distributed architecture. This type of service is offered by Service oriented architecture. Four main approaches for migrating legacy systems to SOA: replacement, wrapping, redevelopment and migration have been highlighted in [18] and the comparison of each approach is done in the area of maturity, applicability, strengths and weaknesses of each of them so that , we can better understand how to choose among strategies for any given project.

Although SOA has been introduced to provide service on the network, it cannot be seen as a replacement to distributed object architecture but as a way of developing loosely coupled distributed systems. They may also be considered superficially similar but there are subtle differences in the two approaches that, taken together, lead to significant differences in terms of their large-scale software engineering properties such as the granularity of service, ease of composition and differentiation – properties that have a significant impact on the design and evolution of enterprise-scale systems[27].They have further emphasized that some features of distributed objects are actually crucial to the integration tasks targeted by service-oriented architectures.

SOA cannot be termed as a replacement but we may say that SOA focuses at a higher level and is considered to relate to large

scale systems and focuses on business functionalities design whereas OO focuses on object design and classification.

Hence, we have briefed up the trend that was adopted for the migration of legacy systems and 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.

4. 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.

4.1 Approaches 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.

4.2 Impact of AOP+LMP in 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.

5. COMPARATIVE ANALYSIS OF AOP AND OOP

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 inheritance 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.

Analyzing the history of legacy software evolution most of the organizations relied on migrating the legacy systems to object oriented framework. The redevelopment was practically inconvenient job and with the explosion of distributed and web based environment, the wrapping approach was intensively used to migrate legacy systems to distributed environment. The introduction of CORBA had eased the transition from mainframe based centralized legacy systems to object oriented distributed systems. The next step was migrating them to web based client server architecture. A number of approaches have already been discussed above. The increasing emphasis on migration of legacy systems to object oriented, component based distributed systems lead to a number of techniques being developed for the purpose.

Another approach to migrating legacy systems that most of the system tend to adopt now are migrating them to service oriented architecture. However, the study [22] based on certain case studies stated that the process of migrating legacy systems into SOA has not always been successful. They highlighted a few success factors such as the potential of legacy systems for being migrated, strategy of migration, SOA governance, the business process of the company, budgeting and resources, legacy architecture, close monitoring and few others on which the adaptation of the legacy systems to the new service oriented architecture is dependent.

Most definitions of SOA make use of web services. However it is possible to implement SOA using any service based technology.

Although the early emphasis of the programmers was to migrate the systems to component based architecture but SOA also has become the next step in software architecture evolution. The similarities have been highlighted by Helmut .Both the architecture seem to have the same goal: To provide a foundation for loosely joined and highly interoperable software architecture, enabling efficient, error-free software development [25].

Nearly all evolution research done in recent years had focused on developing a type of architecture that allows loose coupling and high reusability of its components. That will make the software more efficient, faster, error-free software production.

Therefore, enabling legacy teams to successfully migrate towards object-oriented and component development needs to address a whole range of issues.

On the other hand, Aspect-Oriented Programming is a programming paradigm with deals directly with aspects of concern rather than modules of software code. Therefore, AOP works at different level of abstraction. The purpose of AOP is to remove the tangled code by making it possible to extract the cross cutting concerns from the code and then to combine those aspect with one another and executable code using automating tools. This enables the details of the aspects to be modified without having to modify all software code that the aspects affect.

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 limitations of AOP. AOP is actually doing something that many programmers consider an “Anti Pattern”. The exact pattern is called “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.

6. 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). OOP and AOP are working at different levels of abstraction, OOP at object level whereas AOP at code level. One more constraint is that AOP and software architecture have evolved separately as discipline. Therefore integration of aspects into software architecture is a complex job. The ideas and practices of OOP stay relevant. We have discussed object orientation in the light of both component based development and service oriented architecture. Conceptually, all the approaches define different software system characteristics. However, Aspect-orientation, on the other hand, can be seen as a complementary paradigm affecting the software system on several levels. 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.

7. 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

[14] Massimo Colosimo, Andrea De Lucia, Giuseppe Scanniello, Genoveffa Tortora, "Evaluating legacy system migration technologies through empirical studies", Information and Software Technology (2009) Volume: 51, Issue: 2, Pages: 433-447

[15] G.Canfora, A.R.Fasolino, G.Fratollilo, P.Tramontana, **"**A wrapping approach for migrating legacy system interactive functionalities to Service Oriented Architectures",Journal of Systems and Software,Elsevier Science,2008.

[16] An Object-Oriented Approach to Legacy System Migration Gail Cochrane, TRW Government Information Systems Division, 1996

[17] Nelson H. Weiderman,John K. Bergey,Dennis B. Smith Scott R. Tille "Approaches to Legacy System Evolution",USA,1997.

[18] Asil A. Almonaies, James R. Cordy, and Thomas R. Dean,"Legacy System Evolution toward Service-Oriented Architecture", Canada, 2010.

[19] Eunjoo Lee,Byungjeong Lee, Woochang Shin, Chisu Wu,"Reengineering Process for Migrating from an Object-oriented Legacy System to a Component-based System, Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC'03),2003.

[20] Aniello Cimitile, Andrea De Lucia, Giuseppe Antonio Di Lucca, Anna Rita Fasolino, "Identifying objects in legacy systems using design metrics", The Journal of Systems and Software 44 (1999) 199±211,1999.

[21] Wong Weric, Jenny Li "Redesigning legacy systems into the object-oriented paradigm ",International Journal of Software Engineering and Knowledge Engineering IJSEKE,2004.

[22] Maulahikmah Galinium, Negar Shahbaz, "Factors Affecting Success in Migration of Legacy Systems to Service-Oriented Architecture (SOA) Shared Experiences from Five Case Companies, 2009.

[23] G.Canfora, A.R.Fasolino, G.Fratollilo, P.Tramontana "Migrating interactive legacy systems to Web services", Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference, 2006.

[24] Ying Zou and Kostas A. Kontogiannis "Web-based Legacy System Migration and Integration", 2001.

[25] Helmut Petritsch, "Service-Oriented Architecture (SOA) vs. Component Based Architecture", 2005.

[26] Andrea De De Lucia, Rita Francese, Giuseppe Scanniello, Genoveffa Tortora "Developing legacy system migration methods and tools for technology transfer",2008.

[27] Sean Baker and Simon Dobson Comparing Service-Oriented and Distributed Object Architectures", 2005.

.