Proceedings of the 5th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems, Madrid, Spain, February 15-17, 2006 (pp145-150)

Framework for Evolving Systems

URJASWALA VORA[1], N. L. SARDA[2]
[1]Software Engineering and Operating Systems Department
C-DAC, Mumbai (Formerly NCST)
Juhu, Mumbai-400 049.
INDIA
[2]Department of Computer Science and Engineering
Indian Institute of Technology, Bombay,
Powai, Mumbai-400 076.
INDIA
[1] http://www.cdacmumbai.in
[2] http://www.cse.iitb.ac.in

Abstract: Most real world software systems evolve over time to meet changing requirements or changing business rules. Sometimes, old and new requirements have to be simultaneously met, i.e. old and new rules are needed to exist concurrently. Accommodating more than one rule in the existing entities, leads to the change in the operational system, which involves risk. The evolution in the system impacts the business rules as well as the data model of the system. The evolution step incorporating the change in business rule requires changes to be carried out in the application architecture in terms of addition or modification of process as well as data components and alterations in the relationships among the components. The evolution problem of coexistence of business rules can be considered as incremental change to the existing architectural design of the system to nullify the risk of change to the software. We propose a framework, where temporal aspects of process as well as data components and concurrent validity of multiple business rules, with multiple versions of the components can be effectively implemented. The framework supports evolution without disturbing existing architecture and functionality. This framework is consistent in application to systems irrespective of the design methodology followed for the same.

Key-Words: - Framework, Evolving System, Software Maintenance, Temporal Validity.

1 Introduction

Software Maintenance is a costly activity. As identified by Parikh and Zvegintzov [20], software maintenance consumes 50% of all computer resources and by Boehm [3] that maintenance costs can be up to ten times those of an initial development. The work done by Burd and Munro [9] claims that the costs of the maintenance processes are not distributed evenly across all categories of software maintenance, namely Perfective maintenance, Corrective maintenance, Adaptive maintenance and Preventative maintenance. The study done by Leintz and Swanson [19] shows that 50% of the total maintenance costs are accounted for perfective maintenance, 25% for adaptive maintenance, whereas only 21% of the total costs are attributed to corrective maintenance and 4% for preventive maintenance. Hence we safely can claim that a large part of maintenance costs account for the changes in user requirements and changes in user environment, which include changes in business rules. In today's fast moving world as the frequency of evolving of business rules increases, consequently these maintenance costs also increase.

In this paper we focus on the software evolution, which takes place because of the changes in the business rules. The business rules define functional requirements of the application. The changes in business logic of certain activities result in change in the business rules for the application. The point of focus in this paper is that the new rule can coexist with old rule, i.e. the different rules can have concurrent lifetimes for the same task. Not every evolution in business rules invalidates the existing business rules. These multiple rules existing concurrently for the same activity can be distinguished by specifying their temporal validities.

So we propose an architecture that can deal with multiple rules with specific temporal validities

which are overlapping. At the same time we focus on the need of keeping the existing application architecture undisturbed so that impact of change can practically non-exist as well as maintenance costs due to evolution in business rules can reduce to the considerable extent. The framework is to be developed on top of the application architecture irrespective of the fact that application architecture follows which design methodology, e.g. object-oriented or SSAD. The current related work, which we discuss in the next section, describes the techniques, which provide for the adaptations or evolutions. However, they assume that changes are disjoint, i.e. change will invalidate earlier rule for that activity, whereas, it is quite common in the real world that changes overlap, and also the changed rules are valid overlapping the temporal validities of existing business rules defining particular business activity. This particular need of evolving applications is not considered among the various proposals. In most cases, every change in the application architecture, due to change in the requirements or requirement of additional functionality, is to be handled in unique way depending on the type of change. There is no generic strategy provided to deal with changes to application architecture in general.

In this paper, we present a framework which can be developed on top of the application architecture, so that any change due to evolution need not affect the existing application architecture, at the same time, allows the evolution of the business rule and also accounts for its temporal validity.

We summarize in section 2 the related work in the area of evolving and adaptive systems and point at the differences in the same with respect to our area of focus. Section 3 defines the evolution at the architectural design level. Section 4 has the details of the proposed Framework with the framework components descriptions and the associations between them. Section 5 identifies the capabilities of the Framework. Section 6 concludes the paper by summarizing the contribution of this paper.

2 Related Work

Our partial survey of the existing literature on evolving systems has led us to believe that the existing literature talks about different techniques and methodologies which can help in adapting the existing system by changing the existing software effectively, e.g. Refactoring. But the work acknowledges that some changes have to be done. The literature is talking about changing the architecture if there is any requirement change, but does not account for the temporal validity there. The work though about dealing with evolution at the architectural design level uses the definition of components specific to particular design methodology, e.g. objects in object-oriented methodology, which is not required if the proper abstractions are defined .

Sarda [15] is the basis for the proposal of the Framework and we are proposing the architecture and methodology using the basic framework given in that paper. The paper presents a framework for an application management system as an extension to a temporal database system.

Tokuda and Batory [12] classify architectural evolution under three different modes, schema transformations, the design patterns micro-architectures, and the hot-spot-driven-approach. The paper views the changes as program transformations, which can be automated with object-oriented refactorings. The paper works more on avoidance of hand coding and evolution at architectural level, in above- mentioned modes, with automation done using refactorings.

Foote and Yoder [5], is accounting for system, which evolves itself at runtime. The paper gives the concept of Active Object Model providing meta information about itself so that it can be changed at runtime. Active object models define the objects, their states, the events, and the conditions under which the objects change state.

Dellarocas, Klein and Shrobe [6] propose the concept of "closing the feedback loop" over the entire software evolution process to construct a self-evolving software system. An evolution engine sits alongside a running application, to monitor its execution and to decide when and how to evolve it.
Roberts and Johnson [8] talks about patterns, which can be grouped together and used as a solution, i.e. as a pattern language, for evolving frameworks The patterns are defined for object-oriented architectures only.

Burd and Munro [9] derive a metric whereby the evolution of software can be studied. The metric can be used further for the assessment of the maintainability of code. The selection of a strategy that offers the best overall evolutionary path depends on the assessment of maintenance changes having effect on the comprehensibility of the code.

Cobleigh, Osterweil, Wise and Lerner [10] put forward the concept of Containment Units for recognizing environmental changes and dynamically

reconfiguring software and resource allocations to adapt to the architectural changes.

Yoder and Johnson [11] describe the architectural style for systems needing high flexibility and dynamic runtime configuration. The Adaptive Object-Models, described by the writers has a domain model and domain experts external to the execution of the program can configure rules for its integrity.

Subramanian and Chung in [17] and [18] detail the approach called, NFR Framework, in which software adaptability, as a non-functional requirement (NFR), is treated as a soft-goal to be satisfied (i.e., achieved not absolutely but within acceptable limits). Based on the same approach, Chung [14] lists architectural design patterns, which can be used as potential adaptability enhancers in developing real-time software systems.

Tu and Godfrey [21] present an approach to studying software evolution of long-lived systems that have undergone significant architectural change. The approach provides a query engine and a web-based visualization and navigation interface to help software maintainers in understanding the evolution.

Greenwood et al. [22] recognize the evolutionary need of active architectures. They define the evolution process as combination of process of composition, decomposition and re-composition, expressed in a process-aware Architecture Description Language (ADL).

Minskey [16] states the need of architectural invariants, which have to be set as firewalls between the architectural divisions done to avoid the attacks on the architecture by the software maintainers.

Anderson [1] lists some patterns that help in recording the history of domain objects. The entire collection of patterns reflects the different changes of state.

Carlson, Estepp and Fowler [2] present three patterns showing how to handle objects changing over time with the transparency to client.

3 Evolution in Architectural Design

Software maintenance is defined in IEEE Standard 1219 [IEEE93] as:

"The modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment."

Many times the term evolution is used as substitute for maintenance. We can define software evolution as modification to code and associated documentation due to:

- change in requirements
- need for improvement
- Changes in the business and technology environment

With this, the one mandatory objective is to modify the existing software product while preserving its integrity.

We can represent evolution, E as function on application architecture, A which gives result as evolved architecture, N, i.e. $E(A) \rightarrow N$. Then the types of evolution step we can represent as :

If evolution is incremental, i.e. addition of the component(s), then evolution, E is of type :
$N \rightarrow A + \triangle$, where $\triangle$ is the incremental change in the application architecture.

If evolution step is needed due to the change in the requirement or in the business rule or in the environment, and hence requires the modification of the component(s), then evolution, E is of type :-

Decompose (E) $\rightarrow E^1$

Identify (E^1) $\rightarrow E^2$ where E^2 is subset of E^1 needs to be modified.

Modify (E^2) $\rightarrow E^3$

Compose ($E^3 + (E^1 - E^2)$) $\rightarrow N$

If evolution step needs the coexisting business rules to be taken care of, i.e. existing system is valid but at the same time new business rule need to be accommodated, then evolution, E is of type :-

Decompose (E) $\rightarrow E^1$

Identify (E^1) $\rightarrow E^2$ where E^2 is subset of E1 implementing the business rule which has new rule for a subset of data and has to coexist with the existing business rule for an overlapping time period.

CreateVersion (E^2) $\rightarrow E^{21}$ (1)

TemporalValidity (E^{21}) $\rightarrow T$ (2)

ControlFlowRuleBase (E^{21}) $\rightarrow R$ (3)

Compose ($E^{21} + E$) $\rightarrow N$ (4)

From equations (1) to (4) represent the Proposed Framework by us.

As stated by Cook et. Al. [23] dealing with the evolution at the architectural design stage of the software process is highly desirable, as measurements of the evolvability of the intended system can be made very effectively.

4 The Proposed Framework

The framework consists of following components:

1. Temporal Meta-Data (T)
2. Process Controller (C)
3. Rule Base (R)

4. Archiving Engine (H)
5. Application Architecture (A)

In the Framework, Application Architecture is application specific component and other components (Temporal Meta-Data, Process Controller, Rule Base and Archiving Engine) are generic components, which are common for any system. The Framework can be applied to the systems which are at the ab initio stage of development.

4.1 Temporal MetaData (T)

The Temporal Meta-Data gives the temporal validity periods of all the process components and data components versions in the Application Architecture and also the temporal validities of control flow rules from the RuleBase. The process components we refer here can be of a business process at higher level of abstraction and a module or an object at the lower level of abstraction depending on the design methodology followed. The temporal validity of any component states that the particular component has valid lifetime span of certain 'FromValidTime' to 'ToValidTime'. We can represent Temporal MetaData as

T = { C, fromTime, toTime} where C is the Process or Data Component. When a component is created its toTime = 'F' where 'F' means 'Forever'. When the new version of the component and/or the new control flow rule is introduced the toTime will be updated.

4.2 Process Controller (C)

The Process Controller gets request when a business activity is invoked by the user of the application. The process controller then searches and selects the rule from the Rule Base for that particular activity depending on the activity invoking and the input data. The rule gives the control flow between versions of the components that are required to be executed for the fulfillment of that particular business activity. The component versions are validated against the temporal specifications from the Temporal Meta-Data. Then the correct versions of the components are executed as per the control flow rules for the task to be executed.

4.3 Rule Base (R)

The Rule Base is the set of control flow rules defined for on the versions of components to be used for a particular business activity and for particular input parameters. The Rule Base gets the request from the process controller for the activity invoked, the control flow path of activities followed before the particular activity and the set of input parameters. The rule selected decides the correct flow of control, between the versions of components, to be taken.

R = { ControlFlow(Business Activities) }, i.e. set of rules stating the control flow path among the versions of components, to be taken to achieve a particular task.

4.4 Archiving Engine (H)

The Archiving Engine handles archiving of the Application Architecture component versions and corresponding control flow rules involving those versions of components from the Rule Base, as per the temporal validities defined in the Temporal MetaData. The archiving of these invalid components can be further used in the business analysis of historical data which can support the queries regarding the evolution of business rules over certain period.

4.5 Application Architecture (A)

Application architecture is set of components which can be of type, Data and Process. We are classifying the components at the highest abstraction but that too can vary depending on the design methodology, e.g. object-oriented (OO) systems will have objects consisting the process and data part together at the lower level of abstraction (the higher level of abstraction in OO systems can be achieved by component diagrams as in UML).

4.6 WorkFlow of The Framework

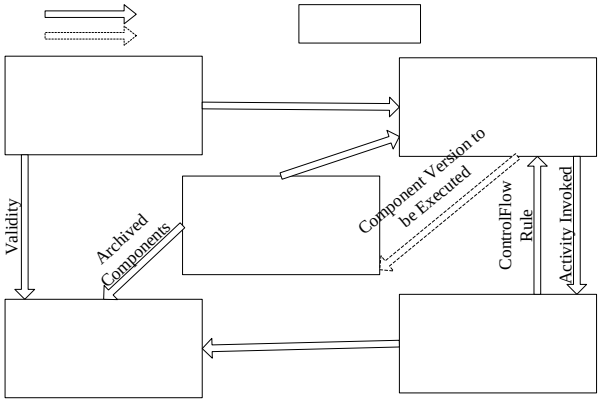
Application Architecture is designed to have the components required by the software system to achieve its business processes. The Rule Base will have the rules for the control flows among these components, for all the activities, which are part of the different business processes. The rule will state if (IP = {….} and InvokingActivity = A) then Invoke(B), where IP is the set of input parameters. The Temporal Meta-Data has the temporal validities of all the process components, which are part of the Application Architecture and also of the data components. The Process Controller will handle the interface between Application Architecture components and the Framework.

With the specifications of temporal validities of the components in Temporal MetaData, the archiving rules are to be specified for the Archiving

Engine so that when the particular business rule becomes invalid the component versions installing the same, are not part of the Application Architecture. The control flow rules involving the component versions to be archived from the Rule Base, are to be archived as well, so that the execution environment does not have the load of invalid component versions and the overhead of searching through the invalid rules.

When evolution in the business logic takes place and new business rule is defined for the business task, which already has one or more valid business rules defined the Framework supports seamless evolution of the system. To implement the new business logic, a version of the component which implements the business rule which has evolved is created. As the temporal validities of the versions of components will be different, i.e. usage period of existing business rule and that of the new business rule which came into existence after evolution of the business logic for the activity, are distinct. This information is captured in Temporal Meta-Data Component of the Framework. When evolution takes place, temporal validities of old versions of components, participating in the task, are modified and temporal validities of the new versions of components are added to Temporal MetaData. The addition of new versions of components to the application architecture results into the new rule, defining the control-flow corresponding to the new versions of components, which are to be added to the Rule Base.

Figure one shows the Framework Components where.



.

5 Framework Capabilities

Here are the evident capabilities of the proposed Framework :

The Framework

- handles the software evolution very effectively without disturbing the existing application architecture.
- accommodates the concurrent existence of business rules and/or data model for a particular business activity provided by the application software with distinct temporal specifications.
- maintains temporal specifications of each and every change in the business rule as well as data model and is capable of producing results for different types of temporal queries which can give answers like “what has changed and when”. These types of queries and their results can help in analyzing the evolution of business processes of the organization.
- can support the statistical analysis of the business processes of the organization, by generating the queries like “what if the change would had varied in …way”.

6 Conclusion

Software evolution is a costly yet unavoidable consequence of a successful application. To accommodate the requirements of today’s fast changing business world, it is required to adapt the new business rules without disturbing the existing working system.

For the requirements of specific kind, where old and new business rules and data models have to exist concurrently in the system, the framework proposed will lead to smooth evolution and will validate the different business rules according to their temporal validities.

References:

[1] Anderson, Francis. A Collection of History Patterns. Collected papers from the PLoP '98 and EuroPLoP '98 Conference, Technical Report #wucs-98-25, Dept. of Computer Science, Washington University, September 1998.

[2] Andy Carlson, Sharon Estepp, Martin Fowler: Temporal Patterns. University Of Washington, Department Of Computer Science, Technical Report TR#WUCS-98-25. http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions.

[3] Boehm B.W., The High Cost of Software, in Horowitz E., Practical Strategies For

Developing Large Software Systems, Addison Wesley, 1975.

[4] Brian Foote and Joseph Yoder. Evolution, Architecture and Metamorphosis. Pattern Languages of Program Design 2, John M. Vlissides, James O. Coplien, and Norman L. Kerth, eds., Addison-Wesley, Reading, MA., 1996.

[5] Brian Foote and Joseph Yoder. Metadata and Active Object-Models. Collected papers from the PLoP '98 and EuroPLoP '98 Conference, Technical Report #wucs-98-25, Dept. of Computer Science.

[6] Chrysanthos Dellarocas, Mark Klein and Howard Shrobe. An Architecture for Constructing Self-Evolving Software Systems. In Proceedings of the Third International Software Architecture Workshop, pages 29–32, Nov. 1998.

[7] D. L. Parnas. Designing Software for Ease of Extension and Contraction. IEEE Transactions on Software Engineering, 5(2): 128-138, March 1979.

[8] Don Roberts and Ralph Johnson. Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks. Pattern Languages of Program Design 3, Robert Martin, Dirk Riehle, and Frank Buschmann, eds., Addison-Wesley, Reading, MA., 1997.

[9] E. Burd and M. Munro. An initial approach towards measuring and characterizing software evolution. In Proceedings of WCRE'99, pages 168-174. IEEE Computer Society, 1999.

[10] Jamieson M. Cobleigh, Leon J. Osterweil, Alexander Wise and Barbara Staudt Lerner. Containment Units: A Hierarchically Composable Architecture for Adaptive Systems. SIGSOFT 2002/FSE-10. Copyright 2002 ACM 1-58113-514-9/02/0011.

[11] Joseph Yoder and Ralph Johnson. The Adaptive Object Model Architectural Style. The Proceeding of The Working IEEE/IFIP Conference on Software Architecture 2002 (WICSA3 '02) World Computer Congress in Montreal 2002, August 2002.

[12] Lance Tokuda and Don Batory. Automating Three Modes of Evolution for Object-Oriented Software Architectures. 5th USENIX Conference on Object-Oriented Technologies and Systems (COOTS '99).

[13] Lance Tokuda and Don Batory. Automated software evolution via design pattern transformations. In Proceedings of the 3rd International Symposium on Applied Corporate Computing, Monterrey, Mexico, 1995

[14] Lawrence Chung. Design Patterns for Adaptable Real-Time Systems. UKC'01, August 10-12, Boston, MA.

[15] N. L. Sarda. A Framework for Application Evolution Management. 10th Australasian Database Conference ADC'99, University of Auckland, New Zealand, 1999 (as part of the Australasian Computer Science Week (ACSW'99))

[16] Naftaly H. Minskey. Towards Architectural Invariants of Evolving Systems. Rutgers University, LCSR, Research Report 1997. http://citeseer.ist.psu.edu/minsky97towards.html.

[17] Nary Subramanian and Lawrence Chung. Software Architecture Adaptability : An NFR Approach. IWPSE 2001. Copyright ACM 2002 158113-508 -4/02/006.

[18] Nary Subramanian and Lawrence Chung. Tool Support for Engineering Adaptability into Software Architecture. IWPSE 2002. Copyright ACM 2002 1-58113-545 -9/02/05.

[19] Leintz B.P., Swanson E.B. Software Maintenance Management. Addison Wesley, 1980.

[20] Parikh G., Zvegintzov N. Tutorial on Software Maintenance. IEEE Computer Society Press, Silver Spring Maryland, 1993.

[21] Qiang Tu and Michael W. Godfrey. An Integrated Approach for Studying Architectural Evolution. 10th International Workshop on Program Comprehension (IWPC'02), 127-136.

[22] R. M. Greenwood , D. Balasubramaniam, S. Cimpan , N.C. Kirby, K. Mickan, R. Morrison, F. Oquendo, I. Robertson, W. Seet, R. Snowdon, B. Warboys, E. Zirintsis. Process Support for Evolving Active Architectures. 9th European Workshop on Software Process Technology, EWSPT 2003, Helsinki, Finland, 2003, pp. 112-127.

[23] Stephen Cook, He Ji, Rachel Harrison: Dynamic and Static Views of Software Evolution. ICSM 2001: 592-601.