Asian Journal of Information Technology 6 (10): 1015-1019, 2007
ISSN: 1682-3915

A Generic Change Propagation Framework to Enhance Service Provisioning in a Grid Environment

O. Ekabua, Obeten, O. Olugbara Oludayo and O. Adigun Matthew
Department of Computer Sciences,
Centre of Excellence for Mobile e-Services for Development, University of Zululand, South Africa

Abstract: Software evolution has piloted from functions, to modules, to objects, to components and now to services. There is a high probability that eventually most software capabilities will be delivered and consumed as services. The basic operation of software evolution is change. Changing services can provide useful information regarding the quality of services provided. But a change can propagate and this can be resolved by further additional changes. These additional changes can also create further propagations, which require further changes until no propagation is left. A propagating change is a ripple effect on service provisioning. In this study, we present a generic change propagation framework to support change automation in a grid environment. Additionally, we discuss change impact analysis and maintenance that form the basis for this framework and propose a change impact analysis factor model to support the framework.

Key words: Change impact analysis, service provisioning, service maintenance, service oriented architecture, grid services

INTRODUCTION

Grid architectures are known for providing flutuating resources, therefore applications that should run in such environments must account for occurable changes. These resources range from processing elements, storage and network resulting from interconnection of parallel machines, clusters or even workstations. Amongst other properties of these resources are their changing characteristics during application execution (Buissen *et al*., 2005). Constructing individual, isolated and dynamic applications that can operate in a distributed environment where resources must be shared, offers a greater challenge. Design issues of diversity, adaptation, data locality, process locality and control locality must be of paramount interest (Edmonds *et al*., 2001).

Changing services can give information regarding the quality of services provided. Change Impact Analysis (CIA) shows what impact or effect a change to a service or service provider will have on the system. It determines the scope of the change and provides a measure of the services complexity. It can be used during service maintenance to keep the system at a high level of quality, avoiding degradation of the system or during development to ensure that quality of the system is maintained throughout the development process.

Service Oriented Architecture (SOA) provides different services ranging from middleware to support service management, data communication, e-healthcare, e-commerce and e-marketing support. SOA is an architectural style whose goal is to achieve loose coupling among interacting software agents. A service is a unit of work done by a service provider to achieve the desired end result for a service consumer. Both the consumer and the provider are roles played by software agents on behalf of their owners. The result of a service is usually a change of state for the consumer, but can also be a change of state for the provider or for both (Hao *et al*., 2003). SOA has also been regarded as services exposed using the Web service protocol suits that include service transportation (such as HTTP, SMTP and FTP), XML messaging (such as XML-RPC, SOAP and REST), Service Description (typically using Web Service Description Language-WSDL, Web Service Endpoint Language-WSEL, Web Services Metadata Exchange-WS-MetaData Exchange and Web Services Dynamic Discovery-WS-Discovery) and Service Discovery (a common registry of services such as the UDDI API, Electronic Business XML-ebXML, Directory Services Markup Language-DSML) (David and Lawrence, 2004).

Corresponding Author: Ekabua Obeten, Faculty of Business, Computing and Information Management, Centre for Systems and Software Engineering, London South Bank University, 103 Borough Road, London SE1 OAA United Kingdom

Service related software applications play important roles in human lives, therefore, products that affect peoples lives must have quality attributes. Good quality software is required and in order to determine the quality of software, we need methods to measure it. A key point to emphasis is that the quality of a service product may change over time and web service related applications are no exception. In the early days of computing, software costs represented a small percentage of the overall cost of a computer-based system. Hence, a sizeable error in estimates of software cost had relatively little impact. But, today software is the most expensive element in many computer-based systems. Therefore, steps taken to reduce the cost of software can make the difference between the profit and loss of a company. So by determining the quality attributes of software, more precise, predictable and repeatable control over the software development process and product will be achieved (Luciano *et al*., 2003).

BACKGROUND INFORMATION

Since software capabilities are being delivered and consumed as services, to improve the quality of software during development, we need models of the development process and within the process we need to select and deploy specific methods and approaches and employ proper tools and technologies. We need measures of the characteristics and quality parameters of the software development process and its stages. We need metrics and quality models to help ensure that the development process is under control to meet the quality objective of the product. Data and measurements are the most basic prerequisites for the improvement and maturity of any scientific or engineering discipline. Yet, in the discipline of software engineering, this area is perhaps one that has many critical problems and one that needs concerted effort for improvement.

The use of measurements, metrics and models in software development assumes the availability of good data. In fact, the poor quality of data is a large obstacle in quality improvement. To enhance data accuracy, a good tracking system for the entire development process must be in place and the system must address the data validation issue. Measurements for software projects, therefore, should be well thought out before being used. Metrics that are arbitrarily established could be harmful to the quality improvement effort of a company and there are numerous examples of this in industry. Each metric used should be subjected to an examination of the basic principles of measurement scale, the operational definition, validity and reliability issues should be well thought out (Kan *et al*., 1994).

As software industry is rapidly moving towards maturity, resources have shifted from being devoted to developing new software systems to making modifications to evolving software systems: Software maintenance. A major problem for developers in a changing environment is that small changes can propagate through software to cause major unintended impacts elsewhere. Therefore, software developers need mechanisms to understand how a change to a software system will impact the rest of the system. This process is called CIA. Making software changes without understanding their effects can obviously leads to unreliable software products. CIA can be used to reduce the amount of maintenance required and thereby increasing the reliability of the software, since fewer errors would have been introduced.

An impact is the effect or impression of one thing on another. Impact can be thought of as the consequence of a change. Impact Analysis (IA) is used to determine the scope of a change request as the basis for accurate resource planning and scheduling and to conform to cost/benefit justification. CIA estimates what would be impacted during service provisioning and related documentation, if proposed service change (sometimes due to a fault or maintenance) is made (Pleeger and Bohner, 1990; Lee, 1998). CIA information can be used for planning changes, making changes and tracing through the effects of changes. Research into CIA has been concerned mostly with procedural software: Function-based programs not serviced-based. SOA is all about services that are found in the real world. CIA is the task through which service providers can assess the extent that a change to a single service will have on the rest of the services. It determines the scope of a change and provides a measure of the services complexity (Li and Henry, 1995). CIA can be achieved by directly finding out the challenges encountered by consumers while receiving services from their clients-service providers.

GRID AND AGENT

The core unifying concept that underlies Grids and Agents systems is that of a service. A service is an entity that provides a capability to a client through a well defined message exchange (Booth *et al*., 2003) or a service is a vehicle by which consumer's need is satisfied according to a negotiated contract, which includes service level agreement and the function offered. In 3rd Generation Grids, all entities are services since service interractions are achieved through web service mechanisms. Although, while every agent is considered a service, not all grid services are necessarily agents. Therefore, the autonomous action notion is a function of how agents and grids interoperate.

The main objective of Grids is that of resource sharing and coordinated problem solving in dynamic, multi-institutional virtual organizations (Foster *et al*., 2001). Grid therefore, provides an infrastructure for federated resource sharing across trust domains. Grid primary concern has been the mechanism by which communities form and operate. Thus, grid effort is devoted to how community standards are represented via explicit policy and enforcement and how actions and commitments by community members are specified, monitored and enforced through implementation.

Agents are autonomous identifiable problem solving entities with well-defined boundaries and interfaces. They are situated in a particular environment, designed to fulfil a specific role and capable of exhibiting a flexible problem solving behaviour in pursuit of their design objectives. They need to be both reactive (reacting to changes in their environment on time) and proactive (taking initiatives) (Foster *et al*., 2004). Agent and Grid systems consist of dynamic and stateful services. Since it is possible for new services to be created and destroyed over the system lifetime, the underlying service model for agents and grids is dynamic (Foster *et al*., 2002).

CHANGE PROPAGATION FRAMEWORK

Changes are endermic to software artefacts and the services provided by these artefacts. When a change is effected in a particular service connected to grid, it is often difficult to determine the propagation of this service changes. We therefore present a change propagation framework shown in Fig. 1 to support change automation in any grid engineering methodology.

The Service Detector Engine (SDE) contains all consumer made available set of grid services (s_1, s_2, . . ., s_n) under utilization. SDE liases with Business Service Bus (BSB), a concept developed by Component Based Development and Integration (CBDI) (David and Lawrence, 2004) and incorporated into our framework to form Service Architecture (SA) responsible for providing a bridge between the implementation and the consuming application, creating a logical view of a set of services, which are available for use and invoke by a common interface and management architecture. The Activity Checker (AC) is responsible for the specification of the constraints that a well-formed service design should satisfy in order to check whether the applications design is in conformance to the main host design. Violation of the rules governing the activity checker will trigger a constraint violation event from the Constraint Activator (CA) to be returned to the Change Propagation Mechanism (CPM). This informs the Service Repairer (SR)

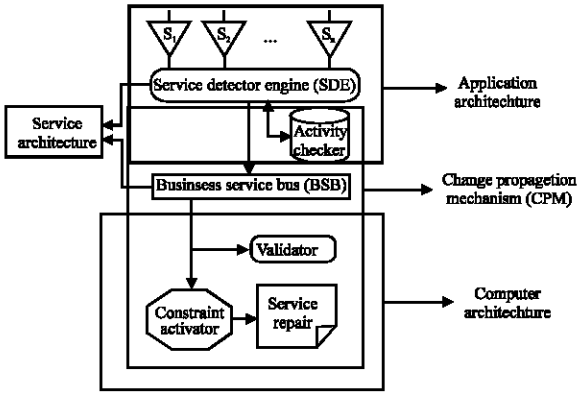

Fig. 1: Generic change propagation framework

of a triggered event calling for a way of fixing the violated constraint by performing actions, which change the application's design and keeps record of the ripple effect. The mechanism Validator (V) is responsible for checking the consistency of the change to the design (through the AC), which can result in further actions.

There are three major architectural perspectives for SOA namely: Application Architecture, Service Architecture and Component Architecture and our framework has these incorporated into it. The architecture has two perspective views: Consumer and Provider. The salient aspect of the architecture is that the consumer of a service should not be interested in implementation detail of a service, but the service provided. This is because the implementation architecture could vary from provider to provider, but still deliver the same service. Additionally, the provider should not be interested in the application that the service is consumed in, because new unforseen application will reuse the same set of services. The consumer's main interest is in the application architecture and the services used, but not in the detail of the component architecture. The interest is in some level of details in the general business objects that are of mutual interest, for example, provider and consumer need to share a view of what is a subscription. But the consumer does not need to know how the service component and database are implemented. Also, the provider is focused on the component architecture and the service architecture, but not on the application architecture. Again, they both need to understand certain information about the basic application inorder to be able to set any sequencing rules including pre and post conditions.

SOA provides the need to be able to manage services as first order deliverables. The communication key between the provider and the consumer is service. There is the need therefore, for a service provisioning architecture in the form of generic framework, that will

ensure that services are not reduced to the status of interfaces, but have an identity of their own and can be managed individually and in sets. BSB as shown in our framework is incorporated to meet this requirement by providing a logical view of the available services for any business domain. BSB answers such questions as what services do I need?, what services are available to me?, what alternative services are available?, what servicecs will operate together and what services are connected to me (Hao, 2003). Present framework is generic because it can be applied to a general service provisioning engineering methodologies to enhance monitoring change propagation. The most important component of the framework is the Change Propagation Mechanism (CPM), which is represented and implemented within the service provisioning architecture and the component architecture. CPM detects any change service due to the triggering effect generated and validated. CPM notifies the SR of the ripple effect for immediate action of fixing the service.

FRAMEWORK IMPLICATION

Maintenance has been recognized as the most costly phase in the software life cycle (Li and Henry, 1995). Since software has been consumed as services, service maintenance effort has been estimated to be frequently more than 50% of the total life cycle cost (Elish and Rine, 2003). This research has the potential to improve service provisioning to customers, thereby cutting cost during service delivery. Using change propagation framework will help to achieve the following:

- Understand the nature of the services needed by a consumer.
- Estimate the effort devoted to a project.
- Determine the quality of service.
- Predict the maintainability of service with respect to the derived benefits.
- Validate best practices for service providers in a frequent changing requirement community.
- Provide optimal maintenance solutions.

By identifying potential impacts before making a change, the risks associated with embarking on a costly change can be reduced, because the cost of unexpected problems generally increases with the lateness of their discovery. The more a particular change causes other changes, the higher the cost. Carrying out CIA will allow an assessment of the cost of the change and help management to choose between alternative changes. It will also allow managers and engineers to evaluate the appropriateness of a proposed modification. If a proposed change has the possibility of impacting large, disjoint sections of a service, the change will need to be re-examined to determine whether a safer change is possible (Foster *et al*., 2001).

Maintenance is costly, difficult and is not always clear what the impact of any type of change to service will have across the whole services. CIA shows the maintainer what the effect of any change will be on the system. This proposed generic framework offers the potential to improve the stability and efficiency of SOA and cut the cost of maintenance.

CHANGE IMPACT ANALYSIS FACTOR ADAPTATION MODEL (CIAFAM)

When a change of service is considered in a system, it is worthy to identify system components that may be impacted after such a change. This would enhance the system to keep running perfectly after a change implementation. A system absorbs a change easily if the impacted components is of a small number. One effective method to account for changes in services is to perform CIA and our framework is accessed by the impact model described. Our main concern is pivoted on how the system reacts to changes that lead to propagation.

For a given change K in a service P, we can describe a set of impacted services as a boolean expression. The Impact Analysis Factor (IAF) for such hypothetical change can be given by (Lee, 1998):

$$IAF(K,P) = A^*(\sim \rho) + A'$$

Where, *, +, ~ denotes the usual boolean operators: conjunction, disjuction and negation, respectively. K = a given change, P = a given service, A = there is an association between K and P, ρ = K is derived from the change service, A' = There is an occurence of aggregation link between K and P and IAF = Impact Analysis Factor. This expression implies that a service in Association (A) with K and not derived ($\sim \rho$) from the change service K or a service that is in Aggregation link (A') with K is impacted. It is important to state that this impact model only predicts, which services would be impacted if a change was really made. If a service is really impacted, it means there is the propensity of propagation in which case the IAF becomes 1. We concentrate on changes that have a synthetic impact, therefore, appropriate measures are based on impact that are dependent on the static nature of the provisioning system. This implies that impacts have a likelihood of propagation (Chaumum *et al*., 1999; Kabail *et al*., 2001).

CONCLUSION

Impact analysis needs to be adapted to the type of systems that become increasingly common today, such as grid-based applications and publish-subscribe systems. The fact that repositories can be shared amongst several distinct systems introduces interoperability dependencies that impact analysis strategies especially tailored for these technologies must address in order to be effective. This study therefore, descibes a generic change propagation framework for automate the effect of changes in a service provisioning environment. Additionally, it identifies the need for CIA technique extensions to service provisioning by the design of CIAFAM.

REFERENCES

Booth, D., H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris and D. Orchard, 2003. Web Services Architecture. W3C, Working Draft http://www.w3.org/TR/2003/WD-ws-arch-20030808/, 2003.

Buisson, J., F. andre and J. Pazat, 2005. Dynamic adaptation for grid computing. Advances in Grid Computing-EGC, LNCS, Springerlink.

Chaumum, M., H. Kabaili, R. Keller and F. Lustman, 1999. A change impact model for changeability assessment in object-oriented software systems. Proceedings of IEEE 3rd European Conference on Software Maintenance and Reengineering.

David, S. and W. Lawrence, 2004. Understanding service-oriented architecture. NET Architecture Centre. Microsoft Architect Journal.

Edmonds, T., S. Hodges and A. Hopper, 2001. Pervasive adaptation for mobile computing. Proceedings of 15th IEEE International Conference on Information Networking.

Elish, M.O. and D. Rine, 2003. Investigation of metrics for object oriented design logical stability. Proceedings of 7th European Conference on Software Maintenance and Reengineering, pp: 193-200.

Foster, I., C. Kesselman and S. Tuecke, 2001. The anatomy of the grid: Enabling scalable virtual organisations. Int. J. Supercomput. Applic., 15: 200-222.

Foster, I., C. Kesselman, J.M. Nick and S. Tuecke, 2002. Grid services for distributed systems integration. IEEE. Comput., 35: 37-46.

Foster, I., C. Kesselman and N. Jennings, 2004. Brain meets brawn: Why grid and agents need each other. Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagents Systems, pp: 8-15.

Hao, H., 2003. What is service-oriented architecture. CTO of SoftTouch Information Technology Pty. webservices.xml.com

Kan, S.H., V.R. Basili and L.N. Shapiro, 1994. Software quality: An overview from the perspective of total quality management. IBM. Syst. J., 33: 1.

Kabaili, H., R.K. Keller and F. Lustman, 2001. A cohesion as changeability indicators in object-oriented systems. Proceedings of IEEE 5th European Conference on Software Maintenance and Reengineering.

Lee, M.L., 1998. Change impact analysis of object-oriented software. Technical Report ISE-TR-99-06, George Mason University.

Li, W. and S. Henry, 1995. An empirical study of maintenance activities in two object-oriented systems. J. Software Maintenance Res. Practice, 7: 131-147.

Luciano, B., H. Reiko, T. Sebastian and V. Daniel, 2003. Modeling and validation of service-oriented architecture: Application Vs. Style. FSEC/FSE. Helsinki. Finland.

Pleeger, S.L. and S.A. Bohner, 1990. A framework for software maintenance metrics. IEEE. Transac. Software Engineering.