academicJournals

Vol. 5(2), pp. 7-14, July, 2014
DOI: 10.5897/IJSTER2014.0247
Aritcle Number: 95AE8A345736
ISSN 2141-6559

http://www.academicjournals.org/IJSTER

International Journal of Science and Technology Educational Research

Review

A software ability network in service oriented Architecture

S. Rajalakshmi

Department of Computer Science and Engineering, SCSVMV University, Kanchipuram, Tamil Nadu, India.

Received 7 March, 2014; Accepted 26 June, 2014

In recent days, Service-Oriented Architecture (SOA) is used as a proficient resolution to integrate and potentially distributed in the software firm and enterprise. Architectures explore great vital role of network evaluation of the system. In a SOA-network value based environment, Pattern proven the solutions and design is one of the most important issues that must be considered because of the loosely coupled nature of SOA. However, there are many functionalities and deal with software Architect services such as flexible, speed, efficiency reliability and so on. SOA brings additional settings of proper governance of design pattern which becomes a critical issue. In this paper, we propose an Architect for Service Oriented Pattern based enterprise can play in transformation terms applying the quality conceptual for framework.

Key words: Service-oriented design, service Intelligence, performance management systems and quality management in SOA.

INTRODUCTION

Software architecture, Hofmeister et al. (1999), intuitively denotes the high level structures of a software system. It can be defined as the set of structures needed to think about the software system, which comprise the software elements, the relations between them, and the properties of both elements and relations. Applying the term “architecture” to software systems is a metaphor that refers to the classical field of the architecture of buildings. Garlan and Shaw, 1993, The term “software architecture” is used to denote three concepts: high level structure of a software system, discipline of creating such a high level structure and documentation Bosch (2004) of this high level structure. Software architecture exhibits the following characteristics: multitude of stakeholders, separation of concerns, quality-driven, recurring styles and conceptual integrity. Software architecture (SA) is considered to be the most importance to the software development lifecycle Outi et al. (2009). It is used to represent and communicate the system structure and behavior to all of its stakeholders with various concerns. SA facilitates stakeholders in understanding design decisions and rationale, further promoting reuse and efficient evolution. One of the major issues in software systems development today is systematic SA restructuring to

E-mail: gomathyck@gmail.com.

accommodate new requirements due to the new market opportunities, technologies, platforms and frameworks.

According to Pressman, Sobiesiak and Yixin (2010) “One goal of software design is to derive an architectural rendering of a system”. Architectural design, detailed design and design reviews provide the most important steps in a cost effective software development process. Software engineering activities are goal directed in order to produce working software in a timely manner within some cost constraints Al Dallal, (2010). For complex computer based systems, software architecture plays a very important role in its success or failure. Software architecture is “the overall structure of the software and the ways in which that structure provides conceptual integrity for a system”. Software architectural design is immensely challenging, strikingly multifaceted, extravagantly domain based, perpetually changing, rarely cost-effective, deceptively ambiguous, and perilously constrained with some exceptions. Service oriented architecture modeling is performed considering various stages of network performing the functionalities and services Xu et al. (2006). This model consists of three stages: architectural analysis, architectural synthesis and architectural evaluation. The model has been extended to include two more stages, implementation and maintenance. All stages are supported by architectural knowledge. The architectural analysis stage serves to define the problems an architect must solve. An architect examines architectural concerns and context in order to come up with a set of architecturally significant requirements.

Another major issue in software systems development today is quality Frigo and Steven (1998). The idea of predicting the quality of a software product from a higher-level design description is not a new one. During recent years, the notion of software architecture has emerged as the appropriate level for dealing with software quality (Rasool and Nadim, 2007). This is because the scientific and industrial communities have recognized that Software Architecture sets the boundaries for the software qualities of the resulting system. The aim of analyzing the architecture of a software system is to predict the quality of a system before it has been built and not to establish precise estimates but the principal effects of architecture (Abdelmoez et al., 2009). Designing architecture so that it achieves its quality attribute requirements is one of the most demanding tasks an architect faces (Taylor et al. 2009). It is demanding for several reasons including lack of specificity in the requirements, shortage of documented knowledge of how to design for a particular quality attributes, and the trade-offs involved in achieving quality attributes (Outi et al., 2009). It would be desirable to have a method that guides the architect so that any design produced by the method will reliably meet its quality attribute requirements.

Literature review

Software architecture provides the solution for which technical and operational problem can be resolve easily. Lots of researchers proposed variety of papers for the given work are given below:

Pradip Peter Dey (2011), presented a strongly adequate software architecture defined along with some other software quality attributes which contributed in formative assessments of software architecture. The architectural categories were not constrained by a particular programming language, or domain. Software engineers have strived for the strongly adequate software architecture. However, software architecting was an iterative process and formative assessments guide that the architects to improve the qualitative aspects in an iterative process. The categories proposed in given paper have intended to help reviewers in formative assessments. The role of formative assessments has stressed during the development process in order to produce revised architectures from initial work or working progress.

Outi et al. (2009) proposed an approach that used SA in software architecture design. A responsibility dependency graph has been given as input and architecture styles and design patterns were used as transformations when searching for a better solution in the neighborhood. The solution was analysed with regard to quality and effectiveness. The experimental results achieved with given approach showed that although extremely high quality values have achieved with given approach, their “true” quality as evaluated by examining the Unified Modeling Language (UML) class diagrams was not actually as good. However, when combining the solution achieved with SA with a GA implementation, the actual quality of the produced solutions increased as well as the calculated metric values. The proposed paper would suggest that further work should be done with studying the combination of these two algorithms in software architecture design. Studying the definition of evaluation functions for simulated annealing and genetic algorithm should be done as well, as using the same function apparently gives quite different types of solutions when using the different algorithms. Their future work attend to these questions as well as deriving real test cases to further evaluate the approach, and adding more design patterns to cover a larger search space of possible architectures. They have planned to implement a multi-objective fitness function primarily for the GA implementation.

Abdelmoez et al. (2009) given a paper in which Software Architecture Risk Assessment (SARA) tool designed and implemented as a tool for computing and

analyzing architectural level risk factors like maintainability-based risk, reliability-based risk and requirement-based risk. By manipulating the data acquired from domain experts and measures obtained from Unified Modeling Language (UML) artifacts, SARA Tool used in the design phase of the software development process to improve the quality of the software product and identify critical components that have high risk levels. They used the product line architecture of a Microwave Oven to demonstrate the usage of SARA tool in assessing PLA. The modified version of the Microwave Model has been aggregated to consist of 9 sequence diagrams and two class diagrams. There were a total of 14 optional and variant classes. From the product line architecture a total of 96 validated product members, were generated with the instantiation process.

Ampatzoglou et al. (2011) suggested a methodology for exploring designs where design patterns have been implemented, through the mathematical formulation of the relation between design pattern characteristics and well known metrics, and the identification of thresholds for which one design becomes more preferable than another. The given approach assisted goal oriented decision making, since it was expected that every design problem demands a specific solution, according to it was special needs with respect to quality and expected size. Their methodology has been used for comparing the quality of systems with and without patterns during their maintenance. Thus, three examples that employ design patterns have been developed, accompanied by alternative designs that solve the same problem. All systems have been extended with respect to their most common axes of change and eleven metric scores have been calculated as functions of extended functionality. The results of the analysis have identified eight cut-off points concerning the Bridge pattern, three cut-off points concerning Abstract Factory and 29 cut-off points concerning Visitor. In addition to that, a tool that calculates the metric scores has been developed.

Christian and Mila (2011) described how component-based systems with multiway cooperation focused on the basis of an architectural constraint that went beyond common acyclicity requirements. The given analysis have concerned on the property of deadlock-freedom of interaction systems and given a polynomial-time checkable condition that ensured deadlock-freedom by exploiting a restriction of the architecture called disjoint circular wait freedom. Roughly speaking, given architectural constraint disallowed any circular waiting situations among the components such that the reason of one waiting was independent from any other one. On the other hand, if their approach failed, the information provided by the entry interactions has given a hint of which components were involved in a potential deadlock. With given information, a software engineer has taken a closer look at given potentially small set of components and either resolve the reason manually or encapsulate given set in a new composite component that has equivalent behaviour, was verified deadlock-free with another technique, and now causes no problems in the remaining system. Their approach used as a design pattern to ensure that a system was correct by construction. If a software engineer sticks to the composition rule imposed by their architectural constraint, a subsequent application of their condition after each composition step facilitated a correct system design in an automatic and convenient way. They concluded the paper with an overview of the current state of affairs in their work on interaction systems. In their research perspective, they followed ideas that ultimately allowed for correctness by construction. They followed the philosophy to develop and investigate design patterns or architectural constraints that were amenable to the formulation of efficiently checkable conditions for the properties in question.

Germán et al. (2010) given a paper in which SAME tool computed the similarity between cases by considering the particular dimensions of connector catalogues. The attributes and values for these dimensions depend upon the overall design context, the application domain and the designer's perspective of the problem. As a consequence, the results of the similarity are function biased. So far, they have taken a simple approach based on the structural characteristics of components playing similar roles when attached to connectors. However, a stronger compatibility check required the components to be also equivalent from a behavioral point of view. A related drawback indicated that there was a lack of behavioral modelling in the C&C architectural specifications. In the current SAME implementation, the designer gives details about the way components behave when interacting with each other's. The proposed method prevented the adaptation of the object-oriented solutions to generate behavioral diagrams - such as sequence diagrams - that provided a more complete picture of the object-oriented implementation to the designer. The behavioral aspect of materializations is a topic for future work. SAME provided an editor for the creation of materialization experiences, the specification of the interaction models was still a highly manual task. To overcome the given situation, they were planning to extend the SAME Eclipse Plugin which provides a user-friendly interface that supported the construction of interaction models for the materialization experiences.

PROBLEM DEFINITION

In present time, software architecture is a major issue in

any software organization, which develops software for some particular organization or firm. Lots of things affect software development life cycle. To design any software designs we have to keep some points in mind to develop effective software in reasonable time and cost. Here we described the following issues, which have to be removed at the time of software design phase (Hofmeister et al. 1999):

What is the most essential part for a software development industry to do to get the main out of its software architects and provide software architectures of the top essential quality?
What should be steps to measure the capability?
In what way the "theory of software architecture competence" look like?
What are the possible organizational practices presently at work to enhance capability?

SOA framework

The desire for enterprise systems that have flexible architectures, detailed designs, implementation agnostic and operate efficiently continues to grow. A major effort towards satisfying this need is to use SOA. Moreover, there is new research and development in order to achieve more demanding capabilities (example, workflow service composition with run-time adaptation to changing Quality of service attributes) that have been proposed for service- based systems, especially in the context of system. A basic concept is for SOA to enable specifying the creation of services that can be automatically composed to deliver desired system dynamics while satisfying multiple Quality of service attributes. As shown in Figure 1. A fundamental SOA concept is to enable flexible composition of independent services in a simple way. The simple concept is crucial since it separates details of how a service is created and how it may be used. This kind of modularity is defined based on the concept of brokers and its realization as the broker service. The SOA conceptual framework lends itself to the separation of concerns ranging from application domains (example, business logic) Information Technology (IT) infrastructure is one of the choices of programming languages and operating systems. The interoperability at the level of services means loose coupling of reusable services. The high-level description of the SOA principals does not account for the operational dynamics of SOA, especially with respect to time-based operations. Therefore, understanding the dynamics of a service-based system using simulation is important. Simulation can also support specific kinds of service-based software systems that are targeted for business processes with specialized domain Knowledge.

SOA resources

Enterprise applications typically require different kinds of interfaces to the data they store and the logic they implement: data loaders, user interfaces, integration gateways and others. Despite their different purposes, these interfaces often need common interactions with the application to access and manipulate its data and invoke its business logic. The interactions may be complex, involving transactions across multiple resources and the coordination of several responses to an action. Encoding the logic of the interactions separately in each interface causes a lot of duplication. As shown in Figure 2. A Service Layer defines an application's boundary and its set of available operations from the perspective of interfacing client layers. It encapsulates the application's business logic, controlling transactions and coordinating responses in the implementation of its operations

SOA architectural model

Service-Oriented Architecture (SOA) has been widely promoted by analysts and IT vendors as the architecture capable of addressing the business needs of modern organizations in a cost-effective and timely manner. Perceived SOA benefits include improved flexibility and alignment between business processes and the supporting enterprise applications, lower integrations costs (in particular for legacy applications), and numerous other advantages. Although, SOA can play an important role in inter enterprise business-to-business (B2B) applications, SOA is primarily regarded as an intra-enterprise architecture used for internal integration. SOA adoption was initially driven by the emergence of Web Services and related technologies and the need to provide a more effective enterprise computing architecture oriented modelling. SOA is explored in network drivers using in service oriented distributed enterprise applications. Service oriented architecture is generally the structure of components in a program or system, their interrelationships, and the principles and design guidelines that control the design and evolution in time. Software engineering, a design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. A design pattern is not a finished design that can be transformed directly into source or machine code. It is a description or template for how to solve a problem that can be used in many different situations. Patterns are formalized best practices that the programmer must implement in the application Object-oriented design. Patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved. Patterns that imply object- orientation or more

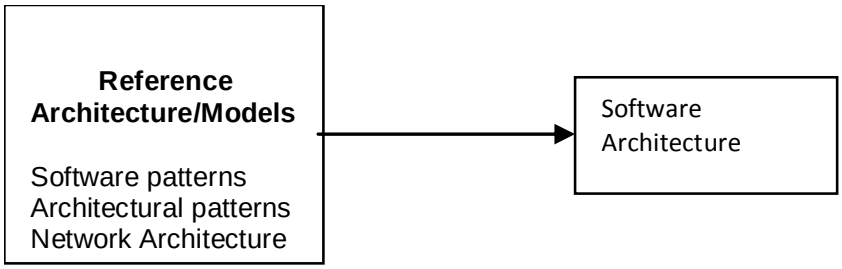

Figure 1. SOA framework model.

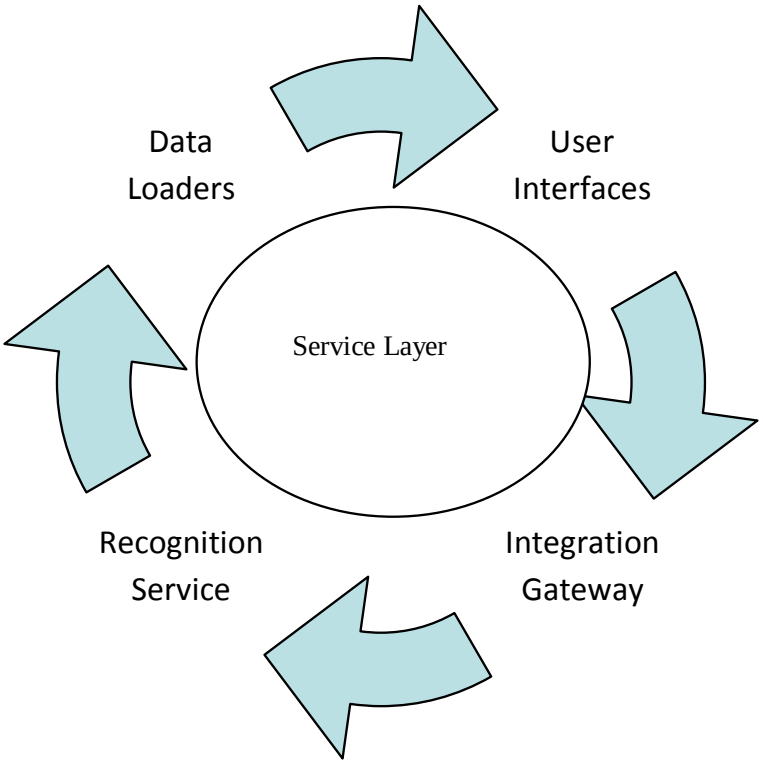

Figure 2. SOA resource activities.

generally mutable state are not as applicable in functional programming languages.

The Software Architect will be responsible for contributing specialized technical knowledge in multiple development efforts using object-oriented analysis and design, Service Oriented Architecture (SOA) and distributed systems. Principle responsibility will be the design and implementation of an enterprise-class platform to enable application supportability and performance management. SOA is the aggregation of components that satisfy a design needs. It comprises components, services and processes. Components are binaries that have a defined interface (usually only one), and a service is a grouping of components (executable programs) to get the job done. This higher level of application development provides a strategic advantage, facilitating more focus on the business requirement. SOA isn't a new approach to soft-ware design; some of the notions behind SOA have been around for years. A service is generally implemented as a coarse-grained, discoverable software entity that exists as a single instance and interacts with applications and other services through a loosely coupled (often asynchronous), message-based communication model. The most important aspect of SOA is that it separates the

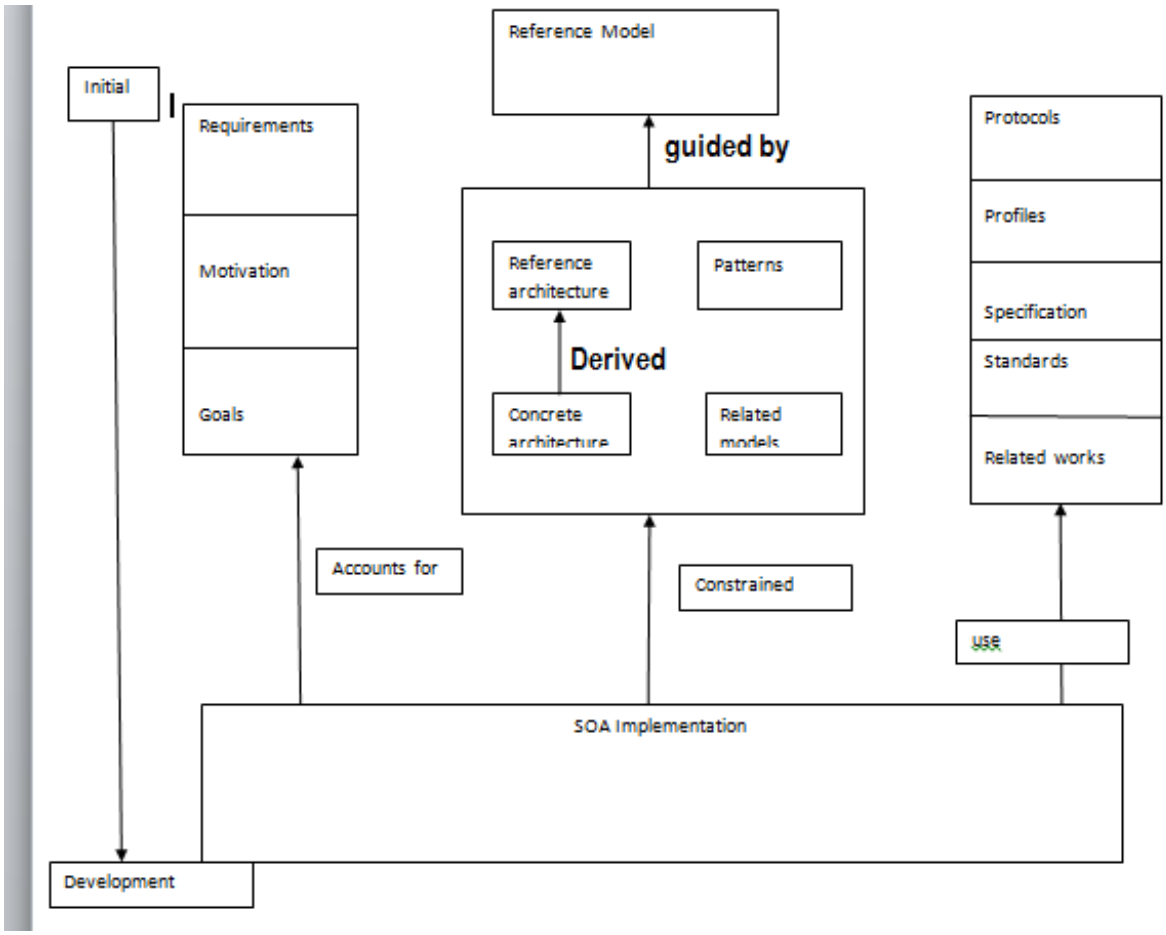

Figure 3. SOA proposed architect design.

service's implementation from its interface. Service consumers view a service simply as a communication endpoint supporting a particular request format or contract as shown in Figure 3.

Reference architecture is a more concrete artifact used by architects. Unlike the reference model, it can introduce additional details and concepts to provide a more complete picture for those who may implement a particular class. Reference architectures declare details that would be in all instances of a certain class, much like an abstract constructor class in programming. Each subsequent architecture designed from the reference architecture would be specialized for a specific set of requirements. Reference architectures often introduce concepts such as cardinality, structure, infrastructure, and other types of binary relationship details. Accordingly, reference models do not have service providers and consumers. If they did, then a reference model would have infrastructure (between the two concrete entities) and it would no longer be a model. The reference model and the reference architecture are intended to be part of a set of guiding artifacts that are used with patterns. Architects can use these artifacts in conjunction with others to compose their own SOA. The concepts and relationships defined by the reference model are intended to be the basis for describing reference architectures that will define more specific categories of SOA designs. Specifically, these specialized architectures will enable solution patterns to solve a particular problem. Concrete architectures may be developed based upon a combination of reference architectures, architectural patterns and additional requirements, including those imposed by technology environments. Architecture is not done in isolation; it must account for the goals, motivation, and requirements that define the actual problems being addressed. While reference architectures can form the basis of classes of solutions, concrete architectures will define specific solution approaches.

Visibility and Real World Effect are also key concepts for SOA. Visibility is the capacity for those with needs and

those with capabilities to be able to see and interact with each other. This is typically implemented by using a common set of protocols, standards, and technologies across service providers and service consumers. For consumers to determine if they can interact with a specific service, Service Descriptions provide declarations of aspects such as functions and technical requirements, related constraints and policies, and mechanisms for access or response. The descriptions must be in a form (or can be transformed to a form) in which their syntax and semantics are widely accessible and understandable. The execution context is the set of specific circumstances surrounding any given interaction with a service and may affect how the service is invoked. Since SOA permits service providers and consumers to interact, it also provides a decision point for any policies and contracts that may be in force. The purpose of using a capability is to realize one or more real world effects. At its core, an interaction is “an act” as opposed to “an object” and the result of an interaction is an effect (or a set/series of effects). Real world effects are, then, couched in terms of changes to this shared state. This may specifically mutate the shared state of data in multiple places within an enterprise and beyond.

The concept of policy also must be applicable to data represented as documents and policies must persist to protect this data far beyond enterprise walls. This requirement is a logical evolution of the “locked file cabinet” model which has failed many IT organizations in recent years. Policies must be able to persist with the data that is involved with services, wherever the data persists. A contract is formed when at least one other party to a service oriented interaction adheres to the policies of another. Service contracts may be either short lived or long lived.

Contribution of the paper

Software architecture is a main concern to improve the experience in current industry for producing quality software at reasonable time and cost. It will examine some of the essential issues, which play an important role in software architecture design and it explored five different phase in organization by which we can provide most essential practices which will be unique models of industry and human behavior that can be given on software architecture design and will be used to help organization and also enhance the architectural capability of personal and organizations.

Phase I: It will analyze the duties, skills and knowledge. We will analyze the work of individuals. In which the skills he/she has and how much knowledge he/she have? We will divide knowledge on the basis on domain specific and technology specific.

Phase II: In this phase we will analyze the human performance technology. It can be measured in the terms of time and cost.

Phase III: In this phase we will analyze the organizational learning. It analyze the learning phase through providing some questionnaires, conducting interviews, identifying change in knowledge and organizational performance.

Phase IV: In this phase we analyze the organizational coordination. In what manner we can provide co-ordination, coordination will be for a team or for some team. The main concern part is generating an inter-team coordination model for firm developing a single product or a closely related set of products.

Phase V: In this phase we will manage the task using neural network. In this phase we will have a group of task using neural network as the main task will be executed.

It will select best task among the group of task. There are number of task an organization has to perform. But the main concern is to know which of the task will be executed first. Choosing the best task according to the environment factors and availability of employees is the best practice in the real world. Software architecture is the set of significant decisions about software of organization which include security, task management, maintainability, performance, resilience, reuse, usability. Our main aim is to enhance these constraints in a proper way. In any organization lots of tasks will be present to perform. Here we will give some priority weightage to each task. In the case of a neural network (NN) based task scheduler, once the job parameters are exactly trained for a particular schedule, it will never miss that given scheduling pattern for that particular task.

CONCLUSION

This paper proposed new intelligence with service oriented architect paradigm to enable system quality to connect with software architectural models from which it is possible to extract precisely information. Our scheme has been proven to have software design with quality in the standard model. A systematic complexity analysis and extensive experiments shows that our proposal is also efficient in terms of computation and design of network used to describe different varieties of messages in SOA.These features of service with network analysis framework scheme a talented solution to group-service oriented communication with access control in various types of design.

Conflict of Interests

The author has not declared any conflict of interest.

REFERENCES

Abdelmoez WM, Jalali AH, Shaik K, Menzies T, Ammar HH (2009). "Using Software Architecture Risk Assessment For Product Line Architectures." In International conference on communication, computer and power. Pp. 15-18.

Bosch J (2004). "Software architecture: The next step." In Software architecture. pp.194-199.

Christian L, Mila MC (2011). “Efficient deadlock analysis of component-based software architectures”, Elsevier.

Frigo M, Steven GJ (1998). "FFTW: An adaptive software architecture for the FFT." In Acoustics, Speech Signal Proc. 3:1381-1384.

Garlan D, Shaw M (1993). "An introduction to software architecture." Adv. Software Eng. Knowledge Eng. pp.1-40.

Germán VJ, Andres DP, Marcelo C (2010). “Reusing design experiences to materialize software architectures into object-oriented designs”, Elsevier.

Hofmeister C, Robert LN, Dilip S.(1999)"Describing software architecture with UML." In Software Architecture pp.145-159.

Outi R, Erkki M, Timo P (2009). "Using simulated annealing for producing software architectures. 4(7):2131-2136. http://www.sis.uta.fi/cs/reports/dsarja/D-2009-2.pdf.

Pradip PD (2011). “Strongly Adequate Software Architecture”. World Academy of Science, Engineering and Technology. 5:12-28. http://waset.org/publications/1338/strongly-adequate-software-architecture.

Rasool G, Nadim A (2007). "Software Architecture Recovery." Int. J. Comput. Inf. Syst. Sci. Eng. 1:3.

Sobiesiak R, Yixin D (2010). "Quantifying software usability through complexity analysis." IBM Design: Papers and Presentations.

Taylor RN, Nenad M, Eric MD (2009). ”Software architecture: foundations, theory, and practice”. Wiley Publishing.

Xu L, Hadar Z, Thomas AA, Debra JR (2006) "An architectural pattern for non-functional dependability requirements." J. Syst. Software 10:1370-1378.