A Brief Survey of Software Architecture Concepts and Service Oriented Architecture

Mohammad Hadi Valipour, Bavar Amirzafari, Khashayar Niki Maleki and Negin Daneshpour
Department of Electrical and Computer Engineering
Shahid Rajaee University
Tehran 1678815811, Iran
Email: {m.h.valipour, b.amirzafari, kh.niki}@sru.ac.ir, daneshpour@aut.ac.ir

***Abstract*—A critical issue in the design and construction of any complex software system is its architecture. Software architecture as an important column of software development process has various methods and roadmaps that all of them have some common principles and inception. Architecture-based approaches have been promoted as a means of controlling the complexity of systems construction and evolution. In this paper we try to describe basics and main structure of software architecture with a conceptual view to this issue. First, architecture definitions are presented. Finally service-oriented architecture (SOA) as one of useful choices for software architecture to develop web software and systems is glossed in a survey.**

***Index Terms*—Software Architecture, Service-Oriented Architecture, Web Services.**

I. Introduction

Software has become increasingly complex, during the history of computing. A number of approaches have been proposed to address the complexity, at different levels, such as "structured programming" [1], and Fred Brooks' idea of "conceptual integrity" [2]. The design phase of the software life cycle has often been divided into high-level design and detailed design. Many concepts in the ordinary course (building) noted that the architecture will be useful to describe the software, which gave birth to the term "software architecture". The concept of software architecture has emerged as design a solution to a high level of the problems of complexity. Brooks wrote in the seventies on the importance of architecture but what is to draw the user interface today, but with a touch of today, the notion of software architecture [2]. Towards the end of 1994, Denning and Dargan proposed "software architecture" for a new software discipline [3], however, the description was similar to a method for development not a definition. Consensus on the term was not made until the first half of the nineties. Garlan and Shaw said in 1996 "explicit attention to the architecture as a separate level of software design is relatively recent" and accordingly their book is subtitled "perspectives on an emerging discipline" [4]. Service-oriented architecture (SOA) is an architecture that modularizes services. You can then coordinate the collection of these services to business processes to life. In a successful SOA, you can recombine these services in various forms for the implementation of new or improved business processes. SOA is a descendant of the logical evolution of the software modularization techniques that go back more than 50 years with the introduction of structured programming. SOA novelty is that it gives you more flexibility in choosing the implementation of technologies and locations for service providers and consumers. The abstract service interfaces also allow suppliers and consumers to evolve independently as long as the interfaces remain stable [5]. The benefits of SOA resulted primarily from a single feature: the stability of the interface service. This stability, compared to the total percentage of systems change, isolates service to consumers in the development of implementation services. This isolation limits the scope of change and the cost of subsequent amendments. To borrow a lot more advantage if you are able to reuse services as they are. Reuse avoids the cost of re-implementation or modification of the functionality encapsulated in the service [5].

II. Software Architecture

A. Common Ideas and Definitions

Software architecture is a "first-cut" at designing the system and solving the problem or fitting the need. What is truly meant by software architecture is that it is actually an asset of plans which guide the selection, construction, modification and perfect use of the enterprise information infrastructure to enable the desired business future state. We use the term "architecture", in contrast to "design", to evoke notions of codification, of abstraction, of standards, of formal training (of software architects), and of style. Architecture is concerned with the selection of architectural elements, their interactions, and the constraints on those elements and their interactions, it also can be viewed a san ad hoc box-and-line drawing of the system that is actually created and intended for solving the problems which are articulated by the specifications. In the mentioned drawing:
- Boxes define the elements or "parts" of the system.
- Lines define the interactions or between the parts.

Necessary to provide a framework in which to satisfy the requirements and serve as a basis for the design. Now what is generally meant by design is that design is concerned with the modularization and detailed interfaces of the design elements, their algorithms and procedures, and the data types needed to support the architecture and to satisfy the requirements [6]. Among its values we can state:
- Current and future vision of the business
- Quality decision making including investment choices

978-1-4244-4520-2/09/$25.00 ©2009 IEEE

- Use of software in a cost effective manner aligned with business operations
- Reduce redundancy
- Reuse information and software components
- Leverage new technology solutions
- Share systems and data
- Integration across the enterprise
- Common standards
- Reduced number of application interfaces
- Identification and planning for missing data

B. The Roles of Software Architecture

While there are numerous definitions of software architecture, at the core of all of them is the notion that the architecture of a system describes its gross structure. This structure illuminates the top level design decisions, including things such as how the system is composed of interacting parts, where the main pathways of interaction, and what are the key properties of the parts. Additionally, an architectural description includes sufficient information to allow high-level analysis and critical appraisal. Software architecture typically plays a key role as a bridge between requirements and implementation (see Figure 1). By providing an abstract description of a system, the architecture exposes certain properties, while hiding others. Software architecture can play an important role in at least six aspects of software development [7].

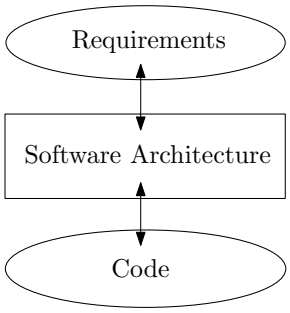

Fig. 1. Software Architecture as a Bridge

- *Understanding*: Software architecture simplifies our ability to comprehend large systems by presenting them at a level of abstraction at which a system's high-level design can be easily understood.

- *Reuse*: Architectural descriptions support reuse at multiple levels. Current work on reuse generally focuses on component libraries. Architectural design supports, in addition, both reuse of large components and also frameworks into which components can be integrated.

- *Construction*: An architectural description provides a partial blueprint for development by indicating the major components and dependencies between them.

- *Evolution*: Software architecture can expose the dimensions along which a system is expected to evolve. By making explicit the "load-bearing walls" of a system, system maintainers can better understand the ramifications of changes, and thereby more accurately estimate costs of modifications.

- *Analysis*: Architectural descriptions provide new opportunities for analysis, including system consistency checking [8], [9], conformance to constraints imposed by an architectural style [10], conformance to quality attributes [11], dependence analysis [12], and domain-specific analysis for architectures built in specific styles [13].

- *Management*: Experience has shown that successful projects view achievement of viable software architecture as a key milestone in an industrial software development process. Critical evaluation of architecture typically leads to a much clearer understanding of requirements, implementation strategies, and potential risks [14].

C. Common Impediments to Achieving Architectural Success

Among the various factors that can lead to unsuccessful architectures there's the lack of sufficient architectural aptitude and/or experience, in adequate time spent on architectural design and analysis, inability to document and communicate the architecture appropriately, and to realize that" standards are not a substitute for a software architecture" to fully comprehend the direct relation between architecture and implementation, to update the documentation as the architecture evolves and etc [15].

III. Service Oriented Architecture

SOA is a design for linking business and computational resources (principally organizations, applications and data) on demand to achieve the desired results for service consumers (which can be end users or other services). OASIS[1] [16] defines SOA as the following:

A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.

Service Oriented Architecture (SOA) is an evolution of the *Component Based Architecture*[2] , *Interface Based Design (Object Oriented)* and *Distributed Systems* of the 1990s, such as DCOM [17], CORBA[3] [18], J2EE[4] [19] and the Internet in general. SOA does not necessarily mean *Web Services* such as *.NET*, *J2EE*, *CORBA* or *ebXML*. These are instead specialized SOA implementations that embody the core aspects of a service-oriented approach to architecture. Each of these implementations extends the basic SOA Reference Model.

[1]The Organization for the Advancement of Structured Information Standards.

[2]A component-based architecture is the model of separating functionality into individual, freestanding objects that can work together.

[3]CORBA is the Object Management Group (OMG) Component Object Reference Broker Architecture. IT is a vendor-neutral, component-based architecture.

[4]Sun Microsystems Java 2 Platform, Enterprise Edition (J2EE) defines the standard for developing component-based, multi-tier, enterprise applications.

While Web services provide support for many of the concepts of SOA, they do not implement all of them. They do not currently support the notion of a contract lease. Also, no official specification provides QoS levels for a service. An organization cannot implement a complete service-oriented architecture given these limitations with Web services. In addition, service consumers can execute Web services directly if they know the service's address and contract. They do not have to go to the registry to obtain this information. Today, in fact, most organizations implement Web services without a registry. Consequently, the extent to which an organization implements an SOA with Web service varies greatly.

A. The Benefits of Service-Oriented Architecture

The main drivers for SOA-based architectures are to facilitate the manageable growth of large- scale enterprise systems, to facilitate Internet-scale provisioning and use of services and to reduce costs in organization to organization cooperation.
The value of SOA is that it provides a simple scalable paradigm for organizing large networks of systems that require interoperability to realize the value inherent in the individual components Indeed, SOA is scalable because it makes the fewest possible assumptions about the network and also minimizes any trust assumptions that are often implicitly made in smaller scale systems An architect using SOA principles is better equipped, therefore, to develop systems that are scalable, evolvable and manageable. It should be easier to decide how to integrate functionality across ownership boundaries. For example, a large company that acquires a smaller company must determine how to integrate the acquired IT infrastructure into its overall IT portfolio. Through this inherent ability to scale and evolve, SOA enables an IT portfolio which is also adaptable to the varied needs of specific problem domain or process architecture. The infrastructure SOA encourages is also more agile and responsive than one built on an exponential number of pair-wise interfaces. Therefore, SOA can also provide a solid foundation for business agility and adaptability.

B. Details of SOA

Before we analyze the details of SOA, it is important to first explore the concept of software architecture, which consists of the software's coarse-grained structures. Software architecture describes the system's components and the way they interact at a high level. These components are not necessarily entity beans or distributed objects. They are abstract modules of software deployed as a unit onto a server with other components. The interactions between components are called connectors. The configuration of components and connectors describes the way a system is structured and behaves, as shown in Figure 2.

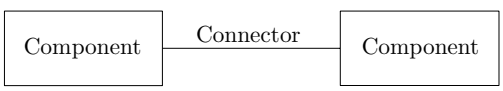

Fig. 2. Software architecture describes a system's components and connectors.

Service-oriented architecture is a special kind of software architecture that has several unique characteristics. It is important for service designers and developers to understand the concepts of SOA, so that they can make the most effective use of Web services in their environment. SOA is a relatively new term, but the term "service" as it relates to a software service has been around since at least the early 1990s, when it was used in Tuxedo to describe "services" and "service processes" [20]. Recently there has been more interest in the software development community about the concepts behind SOA. Figure 3 shows that other technologies can be used to implement service oriented architecture.

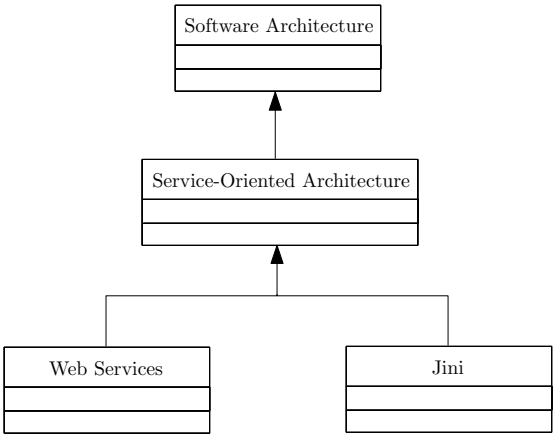

Fig. 3. Web services are one set of technologies for implementing service oriented architecture.

Web services are simply one set of technologies that can be used to implement it successfully.
The most important aspect of service-oriented architecture is that it separates the service's implementation from its interface. In other words, it separates the "what" from the "how". Service consumers view a service simply as an endpoint that supports a particular request format or contract. They are not concerned with how the service goes about executing their requests; they only expect the result.

C. SOA Characteristics

Each system's software architecture reflects the different principles and set of tradeoffs used by the designers. Service-oriented software architecture has these trail characteristics [21]:

1) Discoverable and Dynamically Bound: SOA supports the concept of service discovery. A service consumer that needs a service discovers what service to use based on a set of criteria at runtime. The service consumer asks a registry for a service that fulfills its need.

2) Self-Contained and Modular: Services are self-contained and modular. One of the most important aspects of SOA is the concept of modularity. A service supports a set of interfaces. These interfaces should be cohesive, meaning that they should all relate to each other in the context of a module. The principles of modularity should be adhered to

in designing the services that support an application so that services can easily be aggregated into an application with a few well-known dependencies. Since this is such an important concept when creating services, we will explain some of the principles of modularity and, in particular, how they apply to the creation of services. Bertrand Meyer [22] outlined the following five criteria for determining whether a component is sufficiently modular. These criteria apply equally well when determining whether a service is sufficiently modular.

- *Modular Decomposability*: The modular decomposability of a service refers to the breaking of an application into many smaller modules. Each module is responsible for a single, distinct function within an application. This is sometimes referred to as "top-down design," in which the bigger problems are iteratively decomposed into smaller problems. The main goal of decomposability is reusability. The goal for service design is to identify the smallest unit of software that can be reused in different contexts.

- *Modular Composability*: The modular composability of a service refers to the production of software services that may be freely combined as a whole with other services to produce new systems. Service designers should create services sufficiently independent to reuse in entirely different applications from the ones for which they were originally intended. This is sometimes referred to as bottom-up design.

- *Modular Understandability* : The modular understandability of a service is the ability of a person to understand the function of the service without having any knowledge of other services. For instance, if a banking application implements a checking account service that does not implement a deposit function but instead relies on the client to use a separate deposit service; this would detract from the service's modular understandability.

- *Modular Continuity* : The modular continuity of a service refers to the impact of a change in one service requiring a change in other services or in the consumers of the service. An interface that does not sufficiently hide the implementation details of the service creates a domino effect when changes are needed. It will require changes to other services and applications that use the service when the internal implementation of the service changes. Every service must hide information about its internal design. A service that exposes this information will limit its modular continuity, because an internal design decision is exposed through the interface.

- *Modular Protection* : The modular protection of a service is sufficient if an abnormal condition in the service does not cascade to other services or consumers. For instance, if an error in the checking account service causes invalid data to be stored on a database, this could impact the operation of other services using the same tables for their data. Faults in the operation of a service must not impact the operation of a client or other service or the state of their internal data or otherwise break the contract with service consumers.

3) Interoperability: Service-oriented architecture stresses interoperability, the ability of systems using different platforms and languages to communicate with each other. Each service provides an interface that can be invoked through a connector type. An interoperable connector consists of a protocol and a data format that each of the potential clients of the service understands. Interoperability is achieved by supporting the protocol and data formats of the service's current and potential clients. Techniques for supporting standard protocol and data formats consist of mapping each platform's characteristics and language to a mediating specification. The mediating specification maps between the formats of the interoperable data format to the platform-specific data formats. Sometimes this requires mapping character sets such as ASCII to EBCDIC as well as mapping data types. For instance, a Web service is a mediating specification for communicating between systems. JAX-RPC and JAXM map Java data types to SOAP. Other platforms that support Web services mediate between Web service specifications and their own internal specifications for character sets and data types.

4) Loose Coupling: Coupling refers to the number of dependencies between modules. There are two types of coupling: loose and tight. Loosely coupled modules have a few well known dependencies. Tightly coupled modules have many unknown dependencies. Every software architecture strives to achieve loose coupling between modules. Service-oriented architecture promotes loose coupling between service consumers and service providers and the idea of a few well-known dependencies between consumers and providers.

5) Location Transparency: Location transparency is a key characteristic of service-oriented architecture. Consumers of a service do not know a service's location until they locate it in the registry. The lookup and dynamic binding to a service at runtime allows the service implementation to move from location to location without the client's knowledge. The ability to move services improves service availability and performance. By employing a load balancer that forwards requests to multiple service instances without the service client's knowledge, we can achieve greater availability and performance.

6) Composability: A service's composability is related to its modular structure. Modular structure enables services to be assembled into applications the developer had no notion of when designing the service. Using preexisting, tested services greatly enhances a system's quality and improves its return on investment because of the ease of reuse. A service may be composed in three ways: application composition, service federations, and service orchestration.

- *An application*: is typically an assembly of services, components, and application logic that binds these functions together for a specific purpose.

- *Service federations*: are collections of services managed together in a larger service domain. For example, a checking account service, savings account service, and customer service may be composed into a larger banking-account service.

- *Service orchestration*: is the execution of a single transaction that impacts one or more services in an organization. It is sometimes called a business process. It consists of multiple steps, each of which is a service invocation. If any of the

service invocations fails, the entire transaction should be rolled back to the state that existed before execution of the transaction.
For a service to be composed into a transactional application, federation, or orchestration, the service methods themselves should be subtransactional. That is, they must not perform data commits themselves.

7) Self-Healing: With the size and complexity of modern distributed applications, a system's ability to recover from error is becoming more important. A *self-healing* system is one that has the ability to recover from errors without human intervention during execution.

D. Reliability

Reliability actually measures how well a system performs in the presence of disturbances. In service oriented architecture, services will be up and down from time to time. This is especially true for applications assembled from services from multiple organizations across the Internet. The extent to which a system is self-healing depends on several factors. Reliability depends on the hardware's ability to recover from failure. The network must also allow for the dynamic connection to different systems at runtime. Modern Internet networking protocols inherently provide this capability.

E. Known implementations

While the nature of the services themselves may vary, a common standard for declaring a service is desirable when building an infrastructure. Two such standards exist today: W3C's Web Services Description Language (WSDL) and ebXML's Collaboration Protocol Profile.
Version 2.0 of WSDL is impressive in its completeness and ease of implementation; however, it only covers the basic aspects of service description. ebXML is a joint SOA initiative between UN/CEFACT[5] [23] and OASIS [16] . In addition to providing technical components, the Collaboration Protocol Profile was developed to meet the specific needs of electronic business that involve service-oriented interactions between legal enterprises.

IV. Conclusion

The software architecture is emerging as a discipline over the past ten years [7]. System software architecture describes its coarse-grained structures and properties at a high level. As long as the technology takes care of these structures and properties, the technology can be used for the implementation of the architecture. For example, Jini is a technology that supports service-oriented architecture, because the properties of the SOA. It is important for the application of the concepts of software architecture of a new technology to take full advantage of it. Service-oriented architecture is implemented by Web services and other technologies, but the terms and concepts have recently gained popularity as a result of Web services. For example, the computer industry is the term used for two decades to describe various platforms. Some characteristics of SOA supported by some better than other technologies.

[5]The United Nations Centre for Facilitation of Commerce and Trade (UN/CEFACT) is a globally sanctioned, UN body dedicated to developing trade facilitation and electronic business standards and technologies to be used royalty free by anyone, worldwide.

References

[1] O. J. Dahl, E. W. Dijkstra, and C. A. Hoare, *Structured Programming*. Academic Press, 1972.
[2] F. P. Brooks, *The Mythical Man-Month: Essays On Software Engineering, Anniversary Edition*. Addison-Wesley, 1995.
[3] P. J. Denning and P. A. Dargan, "A discipline of software architecture," *interactions*, vol. 1, no. 1, pp. 55–65, 1994.
[4] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. Prentice- Hall, 1996.
[5] P. Brown, *Implementing soa: total architecture in practice*. Addison-Wesley Professional, 2008.
[6] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT SOFTWARE ENGINEERING NOTES*, vol. 17, no. 4, October 1992.
[7] D. Garlan, "Software architecture: a roadmap," in *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*. New York, NY, USA: ACM, 2000, pp. 91–101.
[8] R. Allen and D. Garlan, "Formalizing architectural connection," in *ICSE '94: Proceedings of the 16th International Conference on Software Engineering*, Sorrento, Italy, May 1994, pp. 71–80.
[9] D. C. Luckham, L. M. Augustin, J. J. Kenny, J. Veera, D. Bryan, and W. Mann, "Specification and analysis of system architecture using rapide," *IEEE Transactions on Software Engineering*, vol. 21, no. 4, pp. 336–355, April 1995.
[10] G. Abowd, R. Allen, and D. Garlan, "Using style to understand descriptions of software architecture," in *SIGSOFT '93: Proceedings of the 1st ACM SIGSOFT symposium on Foundations of software engineering*. New York, NY, USA: ACM, 1993, pp. 9–20.
[11] P. Clements, L. Bass, R. Kazman, and G. Abowd, "Predicting software quality by architecture-level evaluation," in *Proceedings of the Fifth International Conference on Software Quality*, Austin, Texas, October 1995.
[12] J. A. Stafford, D. J. Richardson, and A. L. Wolf, "Aladdin: A tool for architecture-level dependence analysis of software," University of Colorado at Boulder, Technical Report CU-CS-858-98, April 1998.
[13] L. Coglianese and R. Szymanski, "Dssa-adage: An environment for architecture-based avionics development," in *AGARD'93*, 1993.
[14] B. Boehm, P. Bose, E. Horowitz, and M. J. Lee, "Software requirements negotiation and renegotiation aids: A theory-w based spiral approach," in *ICSE '95: Proceedings of the 17th international conference on Software engineering*. New York, NY, USA: ACM, 1995, pp. 243–253.
[15] L. Northrop, "Software architecture and product quality," Presentation for the Boston SPIN, Software Engineering Institute, Carnegie Mellon University, May 2003.
[16] Organization for the advancement of structured information standards. [Online]. Available: http://www.oasis-open.org
[17] Distributed component object model. [Online]. Available: www.microsoft.com/com
[18] Corba basics frequently asked questions. [Online]. Available: www.omg.org/gettingstarted/corbafaq.htm
[19] Java EE at a glance. [Online]. Available: http://java.sun.com/j2ee
[20] P. Herzum, "Web services and service-oriented architectures," Cutter Distributed Enterprise Architecture Advisory Service, Executive Report, 2002.
[21] G. Bieber and J. Carpenter, "Introduction to service-oriented programming," Online whitepaper: www.openwings.org/download/specs, 2001.
[22] B. Meyer, *Object Oriented Software Construction*, 2nd ed., ser. International Series in Computer Science. Prentice-Hall, 1997.
[23] United nations centre for trade facilitation and electronic business. [Online]. Available: http://www.unece.org/cefact