State-of-the-art Web Services Architectural Styles

Chen Wu, Elizabeth Chang

School of Information Systems, Curtin University of Technology, Perth, Western-Australia, 6845, Australia

{Chen.Wu, Elizabeth.Chang}@cbs.curtin.edu.au

Abstract. In this paper, we first identify key architectural properties related to quality aspect of web services architecture. Current web services architectural styles are then summarized. Finally, we give the evaluation analysis with evaluation analysis for web services architect.

1 Introduction

As a promising solution for building distributed systems, web services have increasingly gained significant attention from both industry and academia. While a large number of enterprises are now adopting emerging web services technologies to build their own systems, there is a tremendous difference between a computing system that "works" and one that "works well". We believe such difference, along with many web services adoption challenges [1], is partly attributed to the quality issue of the software systems. Related research shows [4] that software quality can be heavily affected by the systems architecture. Hence in this short paper, we first identify quality architectural properties, and then investigate the state-of-the-art web services architectural styles. Finally we give our evaluation against these identified quality properties.

2 Preliminary Concepts

2.1 Software Quality

According to the IEEE definition [27], software quality is an attributed that indicates: "The degree to which a system, component, or process meets specified requirements". We believe that software quality as an attribute is a significant architectural property which would guide the design of the architectural of software systems.

2.2 Architectural Properties

Architectural properties define the minimum "load-bearing wall" of the architecture design based on system requirements (functional or non-functional, e.g. quality) [2]. Hence any architectural designs goal is to create architecture with a set of architectural properties that is able to cover at least all the basic system requirements. In general, requirements determine system properties, which in turn constrain the architectural design. It is worth noting that non-functional properties are often ignored in architectural design [4]. Nevertheless, we believe it is such non-functional requirements, quality properties in particular, that make tremendous difference between system that "just works" and the one "works well" [5]. Hence in this paper we focus on identifying intrinsic quality-based architectural properties inherent in web services systems.

2.3 Architectural Styles

The use of patterns and styles is pervasive in software discipline to leverage past experience in order to produce better design. Software architectural style encapsulates important decisions about the architectural elements and emphasizes important (not necessarily more) constraints on the elements and their relationships. Hence, "prescriptively, it can be used to constrain the architecture, meanwhile descriptively, exposing certain aspects of the architecture so that violations of those aspects and insensitivity to them will be more prominent" [2].

3 Quality properties

Based on previous related work ([3], [25], and [26]), we identify the following architectural properties for Internet-wide web services applications.

- **Loose-coupling** – A flexible relationship between two or more computer systems that are communicating via data transmission. Loosely-coupled systems work well when either side of the computer systems are subject to frequent changes – such as data elements added, removed, or altered.
- **Interoperability** – The ability of two or more systems or components to exchange information and to use the information that has been exchanged.
- **Scalability** – The capability to support large numbers of web service consumers and providers, or high volume of interactions among these consumers and providers without making major changes to the existing systems.
- **Simplicity** – The architecture does not impose high barriers to entry for its intended adopters: each individual component in this architecture should be substantially less complex as to the extent that they will be easier to understand and implement.
- **Extensibility** – The ability to dynamically accommodate changes (such as adding functionality to a deployed system) without impacting the rest of the system. For instance, the architecture should allow the service consumer to dynamically add

additional information to the message other than what is specified in the shared schema.

- **Performance** – This may include network performance, user-perceived performance, and network efficiency.
- **Security** – The security of Web services across distributed domains and platforms and privacy protection for the consumer of a Web service across multiple domains and services.
- **Reliability** – The degree to which an architecture is susceptible to failure at the system level in the presence of partial failures within components, connectors, or data.
- **Visibility** – The ability of a component to monitor or mediate the interactions between two other components. Visibility can affect several other architectural properties such as performance, reliability and security. For instance, the header of the SOAP can help content routing and security inspection through the gateway or the company firewall.

4 Architectural Styles Review

Existing web services architectural styles can be generally classified into two categories: the ***Broker Style*** and the ***Peer-to-Peer*** Style.

4.1 Broker Style

Current web services architecture is based on the classical broker architectural styles evolving from traditional distributed object technologies. In general, there are two kinds of brokers: *Matchmaker* and *Facilitator* [6]. The matchmaker broker style is described as "publish-find-bind". It is a straightforward approach to distributed computing and it provides the advantage that clients are loosely coupled to the servers only via an interface which can be discovered from the matchmaker broker, i.e. the UDDI registry. Some researchers enhance matchmaker broker style, so that it is able to carry out service selection based on the service behavior (e.g. Quality-of-Service, Non-Functional requirements) rather than simple service function [7, 8, 9]. All these work, in an implicitly or explicitly manner, add a middle layer to the existing matchmaker broker in order to augment the insufficient support from existing registry – UDDI to perform service discovery based on service behavior (e.g. QoS). We use ***Layered Matchmaker Broker*** (LMB) style to describe such architectural design. The provider and consumer brokers work as a "gateway" to encapsulate the request by adding or removing some criteria such as QoS or user requirements, sending to the inner layer – the standard UDDI registry. The primary problem with MB and LMB lies in their centralized indexing scheme provided by web services registry – UDDI [10]. Besides, the possible storage of vast numbers of advertisements on centralized registries hinders the timely updates, thus it is questionable whether centralized registries will scale up to the needs of web services [11]. The facilitator brokers delegates all the message request and response between provider and consumer. Some research utilizes facilitator brokers to provide value-added mediation services such as ho-

mogenizing the heterogeneities among different web services [11, 12]. Basic facilitator style can be augmented into ***Layered Facilitator Broker***. In [13], broker layer addressed the issue of heterogeneity when composing distributed web services.

4.2 Peer-to-Peer Style

The most common application of peer-to-peer computing model in web services can be found in service discovery [14, 15, 16]. Each peer represents a provider and/or a consumer. There is no a centralized registry to store the meta-data for all the participated peers, the service discovery relies solely on the capabilities of each peer by leveraging some P2P service discovery algorithms. Such pure P2P style has one critical problem: no existing registries (e.g. UDDI) are leveraged in service discovery, hence the feasibility and compatibility of their research is questionable since they require the complete abolition of existing service discovery mechanisms, which are already well accepted as normative industry standards in practice. Moreover, it introduces more security and trust risk inherent in P2P computing paradigm. An alternative P2P style – ***Matchmaker + P2P Discovery Style*** – has recently been developed to decentralize the conventional centralized UDDI architecture using P2P technology [10, 17, 18] Here service providers register with the local registry, which forms the P2P network registry federation in a decentralized P2P network. If the requested services cannot be found locally, the registry generates a global query delegated to other registries via the registry federation layered on the P2P overlay. There are essentially two styles to approach the decentralized service composition: ***Split Code*** and ***Mobile Code***. Split code style assumes that distributing the execution of processes necessitate the partition of process specification at design-time. During the run-time each local engine only obtains the partial copy of the whole process, and executes it at local site where the invoked service resides [19, 20, 21, 22]. Nevertheless, in mobile code style, the process specification does not specify the concrete service, leaving it determined at runtime. Both process schema and instance are dynamically delivered to the hosts on which the services reside [23]. There are some other research which combines these two styles into the hybrid (***Split Code + Mobile Code***) style [24].

5 Evaluation of Architectural Styles

Here we use a table of style versus architectural properties as the primary tool for comparative analysis. The table values indicate the relative weight that the style for a given row encompasses on a columns property. Thus for each style we assign a symbol against certain property it induces. Minus (-) symbol gathers for negative influences and plus (+) symbols for positive, with plus-minus (+/-) indicating that it depends on some aspect of the detailed requirement and business contexts.

Table 1. Architectural Style Evaluation Table

Style	Derivation	Loosely Coupling	Interoperability	Scalability	Simplicity	Extensibility	Performance	Security	Reliability	Visibility	Composability
Broker	Matchmaker Broker	+	++	-	+	+/-	-	+/-	-	-	--
	Layered Matchmaker Broker	+	+/-	+/-	++	+	+/-	+	+	+	-
	Facilitator Broker	++	++	--	--	++	--	-	--	++	+
	Layered Facilitator Broker	++	++	-	-	++	-	-	-	++	++
P2P	Pure Peer-to-Peer Discovery	+/-	--	++	+/-	+/-	+	--	++	--	--
	Matchmaker + P2P Discovery	+	+	+	+/-	+/-	+	-	++	-	-
	Split Code + P2P Execution	++	++	++	-	-	++	+/-	+	--	++
	Mobile Code + P2P Execution	+/-	+	++	-	+	+	--	+/-	--	+
	(Split + Mobile) Code + P2PE	+	+	++	--	-	++	--	+/-	--	+

6 Conclusions

In this paper, we approach the architectural quality properties for web services. We identify key architectural properties related to quality aspects of web services architecture. Next we summarize web services architectural styles in current literature. Finally, we evaluate these architectural styles against the architectural properties using a table. However, such analysis still lies in the subjective level. We envision a quality-driven approach to analyze, compare and eventually select the most appropriate architecture styles at the quantitative level in an (semi-) automatic manner.

7 References

1. Ciganek, A.P., Haines, M.N., Haseman, W.: Challenges of Adopting Web Services: Experiences from the Financial Industry, Proceedings of the 38th Hawaii International Conference on System Sciences, 2005
2. Perry, D. E. and Wolf, A. L.: Foundations for the Study of Software Architecture, ACM SIGSOFT Software Engineering Notes, 17(4), Oct. 1992, pp. 40–52.
3. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures, PhD Dissertations, University of California, Irvine CA, USA
4. Bosch, J. 2000, Design and use of software architectures – Adopting and evolving a product-line approach, Addison-Wesley 2000, ISBN: 0-201-67494-7
5. Birman, K., Renesse, R., and Vogels, W.: Adding High Availability and Autonomic Behavior to Web Services, Proceedings of the 26th International Conference on Software Engineering (ICSE04)
6. Wong, H.C. and Sycara, K.: A taxonomy of middle-agents for the Internet, Proceedings of the Fourth International Conference on MultiAgent Systems, July, 2000, pp. 465 - 466.
7. Wang, X., Yue, K., Huang, J. Z., Zhou, A.: Service Selection in Dynamic Demand-Driven Web Services, Proceedings of the IEEE International Conference on Web Services 2004

8. Yu, T. & Lin, K.: The Design of QoS Broker Algorithms for QoS-Capable Web Services, Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE04)
9. Degwekar, S., Su, S.Y.W., Lam, H.: Constraint Specification and Processing in Web Services Publication and Discovery, Proceedings of the ICWS 04
10. Papazoglou, M. P., Kr¨amer, B. J., and Yang, J.: Leveraging Web-Services and Peer-to-Peer Networks, Springer-Verlag Berlin Heidelberg 2003.
11. Paolucci, M., Soudry J., Srinivasan, N., and Semantic Sycara, K.: A Broker for OWL-S Web Services, Proceedings of First Internatioinal Web Services Symposium, 22nd - 24th March, 2004
12. Fuchs, M.: Adapting Web Services in a Heterogeneous Environment, Proceedings of the IEEE ICWS04
13. Piers, P., Benevides, M., Mattoso, M.: Mediating Heterogeneous Web Services, Proceedings of the Symposium on Applications and the Internet (SAINT 03)
14. Emekci, F., Sahin, O., Agrawal, D., and Abbadi, A., A Peer-to-Peer Framework for Web Service Discovery with Ranking, Proceedings of the IEEE ICWS04
15. Schmidt, C. and Parashar, M.: A Peer-to-Peer Approach to Web Service Discovery, World Wide Web, 7(2): 211-229, 2004
16. Banaei-Kashani, F., Chen, C-C., Shahabi, C.: WSPDS: Web Services Peer-to-peer Discovery Service, Proceedings of ISWS04, Las Vegas, USA, June 2004, pp 733-743
17. Thaden, U., Siberski, W., and Nejdl, W.: A Semantic Web based Peer-to-Peer Service Registry Network, Technical Report, University of Hanover, Germany, 2003
18. Sivashanmugam, K., Verma, K., and Sheth, A. Discovery of Web Services in a Federated Registry Environment, Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering (SEKE2004): Workshop on Ontology in Action, Banff, Canada, June 21-24, 2004, pp. 490-493
19. Chafle, G., Chandra, S., and Mann, V.: Decentralized Orchestration of Composite Web Services, In Proceedings of World Wide Web, May 17–22, 2004, New York, USA
20. Muth, P., Wodtke, D., Weissenfels, J., Kotz, D.A.: From Centralized Workflow Specification to Distributed Workflow Execution, Journal of Intelligent Information Systems (JIIS), 10(2), 1998
21. Benatallah, B., Dumas, M., Sheng, Q., and Ngu, A.: Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services, Proceedings of the 18th International Conference on Data Engineering (ICDE02), 2002
22. Nanda, M.G., Chandra, S., Sarkar, V.: Decentralizing execution of composite web services, Proceedings of the 19th annual ACM SIGPLAN Conference on Object-oriented programming, systems, languages, and applications, Volume 39, Issue 10, October 2004, Vancouver, British Columbia, Canada
23. Lakhal, N.B., Kobayashi, T., Yokota, H.: THROWS: an architecture for highly available distributed execution of Web services compositions, Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for e-Commerce and e-Government Applications, 2004
24. Schuler, C., Weber, R., Schuldt, H., and Schek, H.: Scalable Peer-to-Peer Process Management — The OSIRIS Approach, Proceedings of the IEEE ICWS04
25. Austin, D., Barbir, A., Ferris, C. and Garg, S.: Web Services Architecture Re-quirements, W3C Working Group Note 11 February 2004, Retrieved: May 5th, 2005, from http://www.w3.org/TR/wsa-reqs
26. MacKenzie, M. & Amand, S.S.: Electronic Business Service Oriented Architec-ture, OASIA Working Draft 047, 20 August 2004, retrieved: May 30th 2005, from http://www.oasis-open.org/committees/ebsoa
27. IEEE 610.12 IEEE Standard Glossary of Software Engineering Terminology