Reference Architectural Styles for Service-Oriented Computing

Tharam S. Dillon, Chen Wu, Elizabeth Chang[1]

Digital Ecosystems and Business Intelligence Institute, Curtin University of Technology, Perth 6845, Australia
http://debii.curtin.edu.au

Abstract. Architecting service-oriented systems is a complex design activity. It involves making trade-offs among a number of interdependent design decisions, which are drawn from a range of concerns by various software stakeholders. In order to achieve effective and efficient SOC design we believe a careful study of architectural styles that can form the reference architecture is important. Hence, this paper provides a study of architectural styles for the reference architecture of SOC-based software systems. We propose a classification scheme for the architecture styles. These architectural styles are extracted from existing research projects and industry practices based on our classification scheme. For all those identified styles, we present an evolution trend driven by engineering principles for Internet-scale systems. As a result, this paper moves the first step towards creating a Reference Architecture that can be utilised to provide sensible guidance on the design of Web services application architecture

Keywords: Service-Oriented Architecture, Web Services, Software Architecture

1 Introduction

The power and flexibility that Service-Oriented Computing (SOC) can offer to system integration are substantial. As the most promising realization of SOC [1], Web services have the potential to enable business-level integration across heterogeneous platforms. Due to its distributed nature, architecting Web services-based SOC applications is not a trivial task. It requires an experienced architect to make trade-offs amongst a number of interdependent design choices, each of which reflects various concerns demanded by numerous stakeholders from different organizations with disparate business goals and IT infrastructure. A recent survey [2] on Web services adoption, for example, shows that quality requirements such as system security, scalability, reliability, flexibility, and performance have become the most important criteria for a company to choose Web services solutions. Many factors influence the software quality, however, most of these quality requirements can be heavily influenced by the software architecture [3, 4]. Hence, a formal study of the fundamental architectures for Web services is necessary to deliver quality-assured SOC systems. Although several fundamental standards and related case studies have been reported for Web

[1] tharam.dillon@cbs.curtin.edu.au

services design, the merit of rigorously architecting Web services applications has only been partially studied. Each quality requirement listed in [2] might lead to different concerns for that architecture design decision resulting in appropriate compromises [3] to suffice all these requirements. Such a compromise, [5], can be achieved through combining related architectural styles. It is essential to reference an array of well-identified Web services architectural styles with their corresponding rationales and business contexts. This paper examines and evaluates the existing Web services architectural styles, which constitute the reference architecture for SOC applications and elicits an appropriate reference architectural style.

2 Preliminary Concepts

A well-accepted definition of software architecture is given in [6]:

"*The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them*".

Research into software architecture indicates that the various concerns inherent in a software architecture can be modelled as different abstract views, which can be further organized into distinct architectural levels [7]. [7] proposed a multi-level architectural model. (1) The Reference Architecture (RA) which captures both domain requirements and infrastructure requirements at the high level abstract level. (2) The Application Architecture (AA) and (3) The Implementation Architecture (IA). Our paper primarily investigates the Reference Architecture for general SOC-based software systems. Furthermore, RUP[2] defines the Reference Architecture as "*a predefined architectural pattern, or set of patterns, possibly partially or completely instantiated, designed, and proven for use in particular business and technical contexts...*"[8]. In this paper, we use the term '*architectural style*' to define a family of Web services systems in terms of a pattern of structural organization. Software architectural style encapsulates important decisions about the architectural elements. This paper uses the definition from [5] for the architectural style: *Definition: an architectural style is a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relation-ships among those elements within any architecture that conforms to that style*. Constraints are often motivated by the application of a software engineering principle as to an aspect of the architectural elements.

3 The Classification Scheme

A classification scheme is presented to categorise the identified architectures styles into different groups as it helps to understand the common features, allows new styles to be added as they are developed, and provides a framework within which the evolu-

[2] Rational Unified Process®

tion or future trend can be envisioned. We have found that most contemporary Web services architecture can be grouped into three basic families: Matchmaker Style, Broker Style, and Peer-to-Peer style. For each family, we present the styles in a sequence where the fundamental style is introduced first and various derived styles are discussed one after another. These derived styles are examined in section 4 – 6. In addition to these three, we also consider two promising "Web-Oriented" Styles.

4 Matchmaker styles

Early Web services architecture is based on matchmaker style, where a matchmaker component is defined as the 'middle agent that stores capabilities advertisements that can then be queried by requesters'[9]. In Web services architecture, a service provider registers with the UDDI registry its capability information and a service consumer contacts the registry to discover this service provider's detail so that it can bind and interact with it. Providers make their services available by publishing their interface and thus advertising their service.

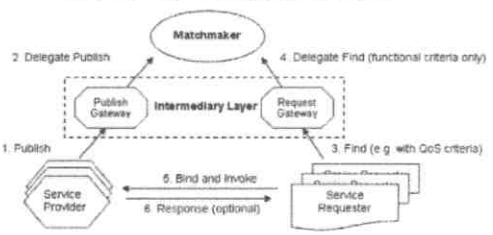

Figure 1 Layered Matchmaker Style

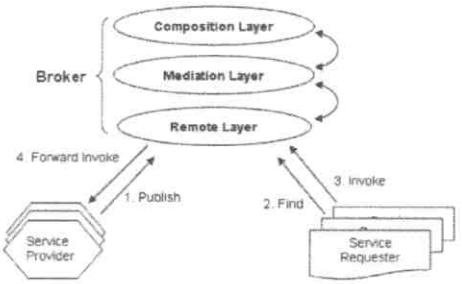

Figure 2 Layered Broker Style

Three classes of matchmaker styles can be distinguished, namely, (1) Layered (LM), (2) Hierarchical (HM), (3) Federated (FM). Service selection based on QoS requirements [10-12] adds an additional architectural layer (see Fig. 1) between the service requester/provider and the matchmaker to collect QoS data, and negotiate Qos requirements between them. Reliance on one single matchmaker can lead to a performance bottleneck and a single point of failure. Research in [13] thus proposed a framework with hierarchical structured registries, each of which maintains a specific business domain.All these registries are managed by one root registry .To address scalability issues, service replication or a federated architecture can be chosen. While [14] stated that "replication was chosen in UDDI because creating a scalable model for distribution of data is inherently difficult", recent researchers have attempted to tackle such distribution issues by introducing a *Federated Matchmaker* style [10, 14, 15].

5 Broker styles

The major difference between a brokers and a matchmaker is that the broker is also involved in the transaction between requester (client) and provider (server).[16] defines a broker architectural pattern (style) as "a distributed software structure with decoupled components that interact by remote service invocation". They specify that the classical broker architectural style includes six major components. The most significant component is the broker component, which distributes client requests to the responsible server components and returns corresponding results.
Four broker-based styles can be distinguished and they are (1) Layered (LB), (2) Asynchronous (AB), (3) Hierarchical (HB), and (4) Federated (FB). LB and AB are illustrated in Figure 2 and 3 respectively. The broker style reduces the complexity involved in developing both service providers and requesters as it makes distribution transparent to the developers [16]. The layered-broker style [17] tackles such a challenge (see Figure 2). The virtual logistics network in [18] provides a real-world example of layered-broker architecture utilised in service-oriented logistics services.

Asynchronous communication provides temporal decoupling, which is crucial for Internet-scaled distributed systems and leads to scalability and resiliency. The *Asynchronous Broker* (see Figure 3) provides a callback mechanism through two Web services standards – the WS-Callback [19] and WS-Addressing [20] . This solves the problem that WS-* specifications have no standard concept for service references. The Publish-Subscribe paradigm [22] is widely accepted as the many-to-many asynchronous communication model. The following three related Web services specifications centre around the topic-based [3] publish-subscribe pattern namely WS-BaseNotification [23] WS-Brokered Notification [24] ,and WS-Topics [25]. Based on WS-Addressing, WS-Eventing [26] provides similar asynchronous capability as does the WS-BaseNotification. Recent real world projects have deployed such an *Asynchronous Broker* style to build in-progress SOC applications such as PSB (Public Services Broker) messaging architecture for e-Government infrastructure.(see Figure 3). One issue with such an Asynchronous Broker is how to match the interests subscribed by service requester with the available notifications published by the service providers. At the time of writing, neither these WS-* specifications nor PSB[27] tackled this issue formatively and thoroughly. WS-Topics partly addresses this issue. The *Triple Space* architectural style, based on the *Asynchronous Broker*, proposes to solve this problem by utilizing semantic web technology. While the Hierarchical Broker style [28] solves the issue of service matching and interaction, and eases the management and complexity of each broker, its structure also brings about a number of shortcomings. Firstly the communication between brokers has to be facilitated by their parent brokers, which limits the flexibility and the velocity of broker interactions. Next, in hierarchical structure, sub-brokers are always controlled by the parent broker, and so are the services controlled by the intermediate broker. This makes it harder to perform dynamic re-organization. The most salient difference between Federated Broker and Hierarchical Broker is the autonomy of the child broker, and thus the flattening of the hierarchical structure. Brokers and services are organized into federa-

tions. Within a federation (a group of services facilitated by a single broker), a service gives up part of its autonomy to the broker.

6. Peer-to-Peer Style

Both matchmaker and broker architectural styles rely on a central control point in contrast to the peer-to-peer (P2P) architectural style. Thus the peer-to-peer Web services architectural style has no centralized registry to store the meta-data of service peers. For this P2P style based web service lifecycle, we discuss service discovery and service composition. P2P based service discovery relies solely on each individual peer's search capability to locate suitable service providers. The first approach to service discovery [29-32] leverages well-established P2P overlay discovery algorithms and places the Web services protocols on top of the native P2P protocols such as Gnutella and DHT [33], with WS-P2P adaptor to bridge the gap between the two protocols. The second approach[15, 34, 35] constructs the P2P communication protocol from the scratch using existing Web services protocols. For instance, [35] presented the PSI model to locate suitable services in a hybrid P2P registry network and the communication engine in each servant forms a Gnutella-compatible P2P network based on the proposed protocol – *probabilistic flooding*. Meanwhile, both of these approaches can also support semantic-based services discovery[29-31, 36].
As indicated earlier, P2P execution (P2PE) is a common means to invoke Web services in the matchmaker style. P2P composition can be classified into three substyles, namely (1) Static, (2) Mobile, (3) Hybrid. .In the Static Composition Style (P2PC-S) style[37] [38] [39], [40], the overall process specification (e.g. BPEL4WS[4]) is, at design-time, partitioned into smaller pieces and deployed to involved service providers and during run-time each local engine only obtains the partial copy of the whole process, and finally executes it at the local site where the invoked service resides. One problem here is that at run-time service providers cannot be changed, thus it will fail to fulfill the dynamic selection of service providers in an unreliable environment. In the Mobile Composition Style (P2PC-M) style[41,42], both the whole process specification and its related instances, which contain the state information of process execution, are dynamically brought to the next invoked service during runtime. [43] employed a combination of *Static Composition Style* to create a true P2P-based service process execution runtime environment and utilized the *Mobil Composition Style* to partition a process into a set of distributed execution units.

[4] Business Process Execution Language for Web Services

7 Web-Oriented Styles

We evaluate two architectural styles that are consistent with Web architectural principles [44]. Representational State Transfer (REST) [5] proponents argue that existing RPC-based Web services has serious weaknesses for the Internet in regards to scalability, performance, flexibility, and implementability[45]. REST specifically introduces numerous architectural constraints to the existing Web services architecture elements in order to: a) **simplify** interactions and compositions between service requesters and providers; b) **leverage** the existing WWW architecture wherever possible. We summarize as follows these constraints which form the fundamental REST-base ('RESTful' Web services) architectural style:

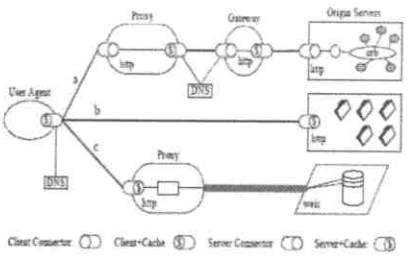

Figure 5 REST Style. Source [5]

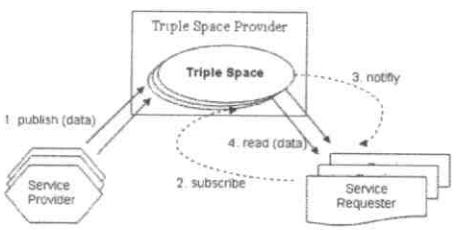

Figure 6 TripleSpace

REST uses a resource identifier (URI) to provide an unambiguous and unique label for one particular web resource. In the RESTful architectural style, all resources are accessed with a generic interface resulting in a dramatic decrease in the complexity of the semantics of the service interface during the service interaction. Choosing these two styles in composing the business process can be found in [47].

7.1 TripleSpace

Triple Space Computing [48] is built on top of of several technologies: Tuple Space[49], Publish-Subscribe paradigm[22], Semantic Web and RDF [50] [46] Triple Space employs the "persistently publish and read" paradigm by leveraging the Tuple Space architecture and APIs. From an architecture perspective,Triple Space is, in effect, based on the natural confluence of *Asynchronous Broker* and *RESTful* styles.. The fundamental interaction among triple space architectural components is shown in Figure 6. The basic interactions between service provider and requester are rather straightforward : The service provider can "***write***" one or more triples in a concrete identified Triple Space. The service requester is able to "***subscribe***" triples that match with a template specified against its interests in a particular concrete Triple Space. Whenever there is an update in the spaces, the Triple Space will "***notify***" related service requesters indicating that there are triples available that match the template specified in its preceding subscription. The notified service requesters "***read***" triples that

match with the template within a particular transaction or the entire concrete space, and further process the triples accordingly. It provides intelligent middleware(broker like), to manage the spaces without requesting each service provider and requester to either download or search through the entire space. Moreover, it needs to provide security and trust while keeping the system scalable and usage simple. Authors in [51] proposed a minimal architecture for such provider middleware. Authors in [48] identified a number of requirements for Triple Spaces (providers): Autonomy (including four basic forms of autonomy: time, location, reference, and data schema), Simplicily, Efficiency, Scalability, Decentralized Architecture, Security and Trust mechanisms, Persistent communications, and History. In order to overcome the lack of support for semantics-aware matching, Triple Space, utilizes RDF to represent and match the machine-processable semantics. It is a promising, if immature, Web services architectural style and may represent the future paradigm for designing and implementing a truly service-oriented architecture.

8 The evolutionary CUBE

Based on previous related work[52-54], we have identified three general architectural design principles in an open environment such as the Internet – *Simplification*, *Decentralization*, and *Loose-coupling*. We believe these three should be equally considered in order to facilitate Internet-scaled Web services computing as they are a crucial prerequisite for any SOC-enabled applications. Each of them acts as an axis in one cubic dimension, which aligns a number of architectural styles in an order that the furthest end reflects the largest positive degree towards that principle. These three dimensions collectively constitute the 'evolutionary cube' as depicted in Figure 7, which provides an overview on current service-oriented computing reference architectural styles.The evolution starts from the *Basic Matchmaker (BM)*, which originates from the widely-accepted 'SOA triangle' architectural style. When both domain and infrastructure requirements become more complicated, the architecture of matchmaker itself becomes more intricate and difficult to design. Even if well designed, such a matchmaker might fail to scale properly in an Internet-wide business context due to its excessive complexity. Hence the appropriate simplification is crucial. The principle of simplification requires the architecture should not impose high barriers to entry for its intended adopters: each individual component in this architecture should be substantially less complex to be easier to understand and implement otherwise functionality of that component needs to be reallocated (by further decomposition or distribution).Under this principle, the *Basic Matchmaker* style moves up along the simplification axis, thus turning into two variant matchmaker styles: *Layered Matchmaker (LM)* and *Hierarchical Matchmaker (HM)*. The consequence of deploying these two variants is to reduce the complexity inherent in each complex matchmaker server, with each one being dedicated in one specific functional area (e.g. remote adaptor, execution, composition, etc.), domain, or geographic area. In other words, the simplification refers to the development and maintenance of each individual server, thus reserving simplicity of core architectural components, while pushing