Design of Service Component Layer in SOA Reference Architecture

Liang-Jie Zhang¹, Jia Zhang²

¹IBM T.J. Watson Research Center, USA

²Department of Computer Science, Northern Illinois University, USA

¹zhanglj@us.ibm.com, ²jiazhang@cs.niu.edu

Abstract

This paper introduces the design of a template of architectural building blocks (ABBs) for the Service Component layer in the SOA Solution Stack (S3) reference architecture. SOA solution architects can use the industry practice-based template as a starting point to quickly configure, customize, and prototype their application-specific service component layer that provides code containers that implement services. We then formalize the issue of service component implementation selection into a one-sided sealed English auction problem. Along with architectural decisions, a cost decision model is deduced to help solution practitioners in selecting proper service component providers for the most cost-effective design of service-oriented solution design.

Keywords: Solution modeling, SOA reference architecture, architectural building blocks, auction model.

1. Introduction

Business applications that are designed and developed using the Service Oriented Architecture (SOA) technology are usually called SOA solutions [1]. To establish a systematic engineering process and methodology to guide SOA practitioners, among others, IBM has published Service-Oriented Solution Stack (S3) model that defines a conceptual view of a service-oriented architecture in the solutioning context [2]. Nine layers are identified and organized into a two-dimensional SOA Reference Architecture. As shown in Figure 1, the horizontal dimension implements functional requirements, and the vertical dimension provides system-support facilities and enablement.

However, these existing models are too high-level to be advisory for enterprise-level SOA solution design and development. They do not provide fine-grained guidance for architects to design reusable and extensible SOA solutions with detailed normative guidance. Our earlier work introduced a set of Architectural Building Blocks (ABBs) as the fundamental unit of the presentation module in an SOA solution [3].

An ABB is a conceptual component that encapsulates internal states and functions and can be configured, extended, and instantiated. An ABB template, comprising a set of ABBs, is identified in each layer in the S3 model. The ABBs collaborate to carry all expected activities. Our

ultimate goal is to build a library of ABBs for all the layers in the S3 model, so that solution architects can use these templates as a starting point, systematically configure and customize the templates, and rapidly prototype their application-specific SOA solutions.

As an example, this paper introduces a set of industry practice-based ABBs for the Service Component layer in the S3 model. A service in the Services layer represents its standard interface that can be accessed by the outer world. The implementation of such a service is maintained in the Services Component layer. Note that a service in the Services layer may have multiple implementations in the Services Component layer. A service in the Service layer may bind to different implementations in the Service Component layer, while users of the service are not aware of the change. Note that practitioners may define their own ABB templates for their own SOA models, based on their own best practice.

Based on our identified ABB template, we establish an auction-based economic model to guide service implementation decision process. We formalize the service component implementation selection issue into a one-sided sealed English auction problem and deduce a cost decision model.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 introduces a set of ABBs for the Service Component layer. Section 4 presents an auction-based service component implementation decision model to support the identified ABB template. Section 5 makes conclusions.

2. Related work

Our work is a continuous effort based on S3. S3 [2] lacks a way to define elements and details internal of any layer. Our previous work introduces a set of ABB templates for the Solution Consumer layer in S3 [3].

Jiang et al. [4] identify parameter-level variation points in WSDL documents and service endpoints. Kim and Doh [5] identify three tiers (presentation tier, service tier, and application tier) in a Web service design and use UML to model variation points in each tier and their relationships. Ruokonen et al. [6] identify a category of variation needs and types relevant for SOA-based systems. In contrast to their work staying at a high level without providing normative guidance for software architects, our work focuses on building an architectural template to help solution architects quickly build a prototype of an SOA

Figure 1. ABBs in Service Component Layer.

solution. In addition, our modeling is based on industry practice-based S3 model that offers a proved guidance for large-scale SOA solution modeling and design.

Model-driven development (MDD) approaches have been widely used in all phases of the lifecycle of software engineering practices to increase software design and development productivity and reduce time-to-market in a systematic manner [7]. Representative examples include OMG's MDA [8], Domain Specific Modeling (DSM) [9], and Software Factories [10] from Microsoft. Different from those component-based modeling approaches focusing on building generic software prototypes, our method focuses on developing architectural building blocks oriented to SOA solution design and development.

Our previous work [11] applies the one-sided sealed auction type in the context of Web services-based auction on the Internet. By contrast, the auction model in this paper is based on a scenario where multiple service component providers bid for an opportunity. Different auction models imply different constraints to the auction problem. For example, our previous auction problem only considers the price; where the auction problem in this paper considers an application-specific profit function based on a set of decision factors. In addition, in contrast to the previous auction model where bidders may show different risk preferences (risk neutral, risk seeking, or risk averse), the bidders (service component providers) considered in this paper are risk seeking.

3. ABBs for Service Component layer

The Service Component layer manifests the IT conformance with each service contract defined in the Service layer; it guarantees the alignment of IT implementation with service description. In detail, each service component fulfills two goals. First, it provides an enforcement point for service realization. Second, it

enables IT flexibility by strengthening the decoupling in the system, by hiding volatile implementation details from service consumers.

3.1 Architectural Building Blocks

Based on industry practice, as shown in Figure 1, we identify eight fundamental Architectural Building Blocks (ABBs) in the Service Component layer: (1) service implementation module, (2) method input transformation, (3) method output adaptation, (4) service deployment module, (5) service publishing module, (6) service invoker, (7) application adaptation module, and (8) service component repository.

A *service implementation module* manages the real implementation (i.e., actual programming code in various languages, e.g., Java, .NET, C++, etc.) of service components in the Service Component layer. A *method input transformation* is responsible for transforming the input parameters of method (i.e., operation) signatures, between the real service components and published service components. A *method output adaptation* is responsible for transforming the output parameters of method (i.e., operation) signatures, between the real service components and published service components. A *service deployment module* is responsible for deploying the actual implementation of a service component to corresponding deployment platforms. This ABB typically manages the deployment descriptor of a service component. A *service publishing module* is responsible for publishing a service component to the Service layer. It directly interacts with the service interaction manager ABB in the Service layer to publish a service component to the service registry ABB in the Service layer. A *service invoker* is responsible for invoking service components (including services from the Services layer or business processes from the Business Process layer). An

application adaptation module is responsible for adapting a bottom-up packaged application from the Operational System layer to the implementation module ABB in this layer, so that it can be treated as a reusable service component. A *service component repository* is responsible for storing information of service components retrievable from (registered with) the Service Component layer. The purpose of this ABB is for reusability. Before creating integration with legacy services, the Service Component layer finds available service components.

3.2 Dependencies and interactions (patterns)

Figure 2 uses a UML component diagram to illustrate a static view of the relationships between the ABBs within the Service Component layer. The service implementation module ABB is the controller of the Service Component layer. All identified ABBs are represented as components in the diagram, with “ABB” as stereo type. The Service Component layer is represented as a package containing all identified ABBs.

Figure 2. ABB relationship diagram.

Figure 3 uses a UML component diagram to illustrate a static view of the relationships for the ABBs across layers. Various layers identified in the S3 are represented as packages; ABBs are represented as components.

To comply with the definition of a service defined in the Services layer, a service component in the Service Component layer must perform any necessary translations to/from XML due to dissimilar information formats that are consumed/emitted some application packages in the Existing Application Asset layer. These translation services are usually provided by the Integration layer.

A service can be realized through the aggregation of existing service components and/or additional behaviors

from the Operational System layer.

IT Governance has a significant influence on the Service Component layer. The choice of implementation technology, the manner in which service components may or may not consume behaviors from other service components and decisions about where to place integration logic, are two examples of where this layer may be influenced by IT Governance.

By its nature, this layer is coupled to the Services layer. A service definition change is likely to cause a direct side-effect on the service component in this layer. For example, if a service is retired from the Services layer, the corresponding service component will also be retired. In addition, it is the responsibility of service components to faithfully reflect the definition of one or more services. To ensure this relationship, a service component must not exhibit behaviors not defined in a service description.

Service components often consume behaviors from the Operational System layer. The relationship creates a dependency on the behaviors consumed. If a decision is made to change the manner in which the latter layer behavior is realized, there are likely side-effects on the service components that consume it.

4. Service component implementation decision Model

4.1 Decision considerations

As shown in Figure 1, the service implementation module decides an actual realization method for a service component. Using the SOA concept, such a design decision may face several options. For example, a service component may be implemented by an in-house project building from scratch, an adaptation from an off-the-shelf service, or a third-party organization through an outsourcing model.

Such a design decision typically has to take into consideration of functional or non-functional aspects. The former aspect refers to architectural design decisions of a service component based on required functions and infrastructures. For example, because of the underlying platform environment, a service component may require to be implemented using the Java technique.

The latter aspect refers to quality-related decisions regarding a service component. Our industry practice

Figure 3. ABBs in the Service Component layer across layers.

summarizes four key decision factors: skill, resource, location, and profit. Skill refers to application-specific required skill sets. Resource refers to physical and non-physical requirements (e.g., required time frame). Location refers to specific location requirements, e.g., only US companies may compete. Profit refers to how to decide an approach for the maximum benefit with the minimum cost.

Functional aspects-based decision typically relies on solution architects based on their technical expertise and experience, which will be discussed in more detail in Section 4.3. As discussed above, non-functional aspects-based decision is more subtle. Especially, its comprising profit factor may subject to negotiations with potential service component providers. Therefore, we formalize the profit decision issue into an auction problem and discuss the decision strategy.

4.2 A Formalized Profit Model

The actual profit of adopting an implementation approach may be decided by a function over a set of attributes in addition to a single price attribute. Although different applications may require specific attribute considerations, our industry practice shows that some key attributes are common. Consider the actual profit value of a specific service component provider (bi):

$profit(b_i) = f(c_i, q_i, t_i, r_i)$, where:

- c_i refers to the cost of the service component proposed by b_i ;
- q_i refers to the quality of the service component proposed by b_i ;
- t_i refers to the required time frame of the service component proposed by b_i ;
- r_i refers to the relationships between b_i and the service component requestor that may lead to some benefits like discount.

The above heuristic function shows what a service component requestor (i.e., service implementation module) can actually obtain from adopting a specific service component from a provider. However, usually service component providers are willing to negotiate.

To gain the highest benefit especially under tough economic situation, solution architects may examine multiple third-party organizations before making a decision. In this selection process, theoretically any traditional business negotiation approaches could be used for bargaining. As one of the fundamental business operation approaches, auction can be used to help in selecting the most cost-effective service component provider.

Auction has been widely used as an important

bargaining form in human society [12]. An auction activity typically includes one or more identical auction items, one or more sellers, and one or more bidders. Bidders compete for the auction items; and the bidders bidding for the highest prices will be granted the auction items.

Applying the format of auction to service component implementation selection, the service implementation module (auctioneer) acts as a seller for opportunity; multiple service component providers bid for the opportunity. A service component provider (bidder) who bids the highest wins the opportunity. Without losing generality, we follow the way of traditional auction research to consider only one bidding object (opportunity) in such an auction. As shown in this specification, the form of auction fits well in the service component implementation negotiation context.

To automate the decision process, we first define a generic service component implementation auction problem.

Definition 1. A service component implementation auction problem can be defined as a 4-tuple $SCIA = (sc, SCP, R_c, R_s)$ where:

- $sc = (spec, sc_{SLA}, sr, f)$ is a 4-tuple that represents the service component implementation opportunity to be bid on, where:
 - $spec$ is the functional description of the service component;
 - sc_{SLA} is the service level agreement (non-functional requirements) associated with the service component;
 - sr is the service component requestor who acts as the *seller* in the action;
 - f is the profit calculation function for the opportunity.
- $SCP = \{sp_1, sp_2, \dots, sp_m\}$ is the set of third-party companies or various development teams within one enterprise who acting as *bidders* for the opportunity to implement the service component. In other words, the service component requestor may face multiple bidders.
- R_c is the common set of rules that governs the service component implementation auction, and
- R_s is a set of rules that is defined for the specific service component implementation auction and governs the auction.

Definition 2. The service component implementation decision implies the following three fundamental rules:

- r_1 (Private value): The private information of a service component provider is its own value for the bid, and it does not depend on what other providers know.
- r_2 (Independent service component provider): All

service component providers are independent in terms of what they can provide, when and where they find the auction, whether to participate in the auction, and what to bid for the opportunity. In addition, there is no communication between them.

- r_3 (Symmetry): Service component provider's private value comes from the same distribution function that is commonly known by all service component providers and the corresponding service component requestor (opportunity seller).

4.3 Selection strategy

There are various types of taxonomy [13] used in the traditional auctions regarding auction seller strategies. Some auction strategies are symmetric to others, such as English auctions and Dutch auctions. Some auction strategies are extensions to others, such as one-price outcry auctions and all-price outcry auctions. Therefore, we decide to investigate English auction based on single item-sealed type, meaning that auctioneers start at a low profit value and incrementally increase the price; only bids are permitted and sellers are not allowed to "ask." In addition, whether a service component provider allows itself to cancel an auction in the process is rather related to its business choice than a scientific decision. Therefore, we do not consider the marketable/non-marketable auctions.

Economists have been studying auction-related behaviors and decision making process and their research results are summarized in the well-known framework game theory [14]. According to the game theory, sellers need to predict the behaviors of the bidders to expect their profits.

Definition 3. In a service component implementation auction, a service component provider chooses a strategy x . It bears a von Neumann-Morgenstern (vNM) utility function U over x as $U(x)$ and a mathematical mean function E over x as $E[x]$. The provider x is risk averse if $E[U(x)] < U(E[x])$; risk neutral if $E[U(x)] = U(E[x])$; and risk proclivity (or, risk-loving or risk-seeking) if $E[U(x)] > U(E[x])$.

In an auction, a seller cares most the bidders' risk preferences, that is, whether the bidders truly want to win the auction or not. A seller could adjust its auction strategy to gain the most profit, by predicting the bidders' risk preferences. In the context of a service component implementation auction, all service component providers aim to win the bid. In other words, service component providers can be viewed as risk seekers.

Definition 4. In the process of a service component implementation auction, each service component provider

scp_i specifies five values $(e_i, l_i, h_i, b_i, B_i)$ on bid:

- $e_i \in R$ represents the private value that the service component provider holds for the opportunity;
- $l_i \in R$ represents the lowest value that the service component provider would offer for the opportunity;
- $h_i \in R$ represents the highest value that the service component provider would offer for the opportunity;
- $b_i \in R$ represents the service component provider's bidding offer during the auction; and
- $B_i \in R$ represents the value that the service component provider finally provides for the opportunity. The winner provides the value of B_i ; others provide nothing.

Therefore, a service component provider scp_i 's utility function u_i is defined as a function over its risk preference rp_i : $u_i = f(rp_i)$. u_i shows a non-linear function yielded by risk-seeking preference:

$$u_i = (e_i - B_i)^\theta, \theta \in (1, \infty), \text{ where:}$$

e_i is the private value service component provider scp_i assigns to the opportunity, B_i is the final offer of scp_i . In other words, if the service component provider scp_i wins the bid, it will gain the value of $e_i - B_i$, and 0 otherwise.

Based on the functional form defined above, we will come to the symmetric Nash equilibrium strategy that a service component provider has for bidding [14]:

$$b_i(e_i) = \frac{p-1}{p-1+\theta} e_i$$

Our previous work [11] shows that the first-price strategy is more profitable for sellers facing risk-seeking bidders. Thus, the service component requestor's highest expected revenue can be deduced as follows:

$$\begin{aligned} r_{\pi 3} &= E[\text{winner's offer}] = E[b_i | i \text{ wins the bid}] \\ &= \frac{p-1}{p-1+\theta} P(i \text{ wins}) E[b_i | b_i \geq b_j, \forall i \neq j] \\ &= \frac{p-1}{p-1+\theta} P(t_i \geq t_j) E[b_i | b_i \geq b_j, \forall i \neq j] \\ &= \frac{p-1}{p-1+\theta} \int_0^1 [F(u)]^{p-1} u f(u) p du \\ &= \frac{(p-1)p}{(p+1)(p-1+\theta)} \end{aligned}$$

4.4 Architectural Decisions

In addition to economics-based service component implementation selection, technical considerations are

typically required. Several technology alternatives exist for the implementation of the service components. The selection criteria should be a balance of the following four criteria: capability, familiarity, strategic, and manageability.

Capability means that the capability to realize the value proposition of service components and to realize the required behaviors of a given service. Familiarity means whether the capability of using the technology already exists in the organization. Strategic means whether the technology inline with the technology roadmap of the organization. Manageability means whether the technology allows for the effective management of service components with respect to the KPIs defined.

Often the selection requires prioritization of these criteria. One alternative may offer features that are suited to the needs of a particular service component (e.g. message mediation) but not offer the management capabilities that other service components require (e.g., high availability). We summarize some common architectural decisions for service component selection:

- Should integration always occur through an integration broker? Can it be performed inside a service component?
- Should collaborating service components consume each other through the Services layer or through a platform-specific interface? (e.g., EJB to EJB or EJB to service)
- Where are transformations performed? In service components or in the Integration layer?
- Should component implementations be portable?

5. Conclusions

In this paper, we have introduced a set of architectural building blocks for the Service Component layer in the S3 reference architecture. We have also presented an auction-based service component implementation decision model to help solution architects in selecting service component providers for the most cost-effective design. Currently, our prototype modeling innovation is part of the IBM SOMA Modeling Environment (SOMA-ME) [15].

We have identified several future research directions. First, we plan to establish a formal model reflecting the actual profit of adopting a service component. Second, we plan to build a formal analytical approach based on a mathematical model, to help solution architects evaluate their model-based ABB customization and configuration and monitor dynamic behaviors of ABBs and their relationships. Third, we plan to accumulate industry practice data to create benchmarks for the presented profit model in this paper.

6. References

[1] L.-J. Zhang, J. Zhang, and H. Cai, *Services Computing*. Springer, 2007.

[2] A. Arsanjani, L.-J. Zhang, M. Ellis, A. Allam, and K. Channabasavaiah, "S3: A Service-Oriented Reference Architecture", *IT Professional*, May, 2007: pp. 10-17.

[3] L.-J. Zhang, J. Zhang, and A. Allam, "A Method and Case Study of Designing Presentation Module in an SOA-based Solution Using Configurable Architectural Building Blocks (ABBs)", in *Proceedings of the 2008 IEEE International Conference on Services Computing (SCC 2008)*, Jul. 8-11, 2008, Honolulu, HI, USA, pp. 459-467.

[4] J. Jiang, A. Ruokonen, and T. Systä, "Pattern-Based Variability Management in Web Service Development", in *Proceedings of 3rd IEEE European Conference on Web Services (ECOWS)*, Nov. 14-16, 2005, Växjö, Sweden, pp. 83-94.

[5] Y. Kim and K.-G. Doh, "Adaptable Web Services Modeling using Variability Analysis", in *Proceedings of 3rd International Conference on Convergence and Hybrid Information Technology (ICCIT)*, Nov. 11-13, 2008, Busan, South Korea, pp. 700-705.

[6] A. Ruokonen, V. Räisänen, M. Siikarla, K. Koskimies, and T. Systä, "Variation Needs in Service-Based Systems", in *Proceedings of 2008 6th European Conference on Web Services (ECOWS)*, Nov. 12-14, 2008, Dublin, Ireland, pp. 115-124.

[7] M.J. Escalona and G. Aragón, "NDT: A Model-Driven Approach for Web Requirements", *IEEE Transactions on Software Engineering*, May, 2008: pp. 377-390.

[8] OMG, "Mda Guide Version 1.0.1", Jan. 6, 2003, Available.

[9] G. Karsai, J. Sztipanovits, A. Ledeczki, and T. Bapty, "Model-Integrated Development of Embedded Software", *Proceedings of the IEEE*, 2003, 91(1): pp. 145-164.

[10] J. Greenfield and K. Short, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley Pub., Indianapolis, IN, USA, 2004.

[11] J. Zhang, N. Zhang, and L.-J. Zhang, 2008., *An SLA-Based Auction Pricing Method Supporting Web Services Provisioning*, in book chapter of "Web Services Research and Practices". 2008, IGI Global, Hershey, PA, USA.

[12] V. Krishna, *Auction Theory*. Academic Press Inc.(London) Ltd., London, UK, 2002.

[13] P. Klemperer., "Auction Theory: A Guide to the Literature", *Journal of Economic Surveys*, 1999, 13(3): pp. 227-286.

[14] A. Mas-Colell, M. Whinston, and J. Green, *Microeconomic Theory*. Oxford University Press, 1995.

[15] L.-J. Zhang, N. Zhou, Y.-M. Chee, A. Jalaldeen, K. Ponnalagu, R.R. Sindhgatta, A. Arsanjani, and F. Bernardini, "SOMA-ME: A Platform for the Model-Driven Design of SOA Solutions", *IBM Systems Journal*, 2008, 47(3): pp. 397-413.