A Multi-Criteria Web Services Composition Problem

Buhwan Jeong and Hyunbo Cho
Department of Industrial and Management Engineering
Pohang University of Science and Technology (POSTECH)
San 31 Hyoja Pohang 290-784 South Korea
{bjeong, hcho}@postech.ac.kr

Boonserm Kulvatunyou and Albert Jones
Manufacturing Systems Integration Division
National Institute of Standards and Technology (NIST)
100 Bureau Dr. Gaithersburg MD 20899
{serm, albert.jones}@nist.gov

Abstract

With its prevalence in enterprise applications integration, the service-oriented approach has been studied in various ways. The popularity, however, results in a number of different standards and implementations. The approach needs agreed-upon definitions and assumptions. To this end, the paper presents a multi-criteria service composition problem and positions to state service engineering functions associated with the problem in a regular expression. The paper also introduces an information compatibility concept to measure the degree of interoperable data exchanges between services.

1. Introduction

Since its introduction, the service-oriented approach has been prevailing in deploying enterprise applications and integrating them within and across organizational boundaries. A public registry publishes a number of various services provided by anonymous developers and any integrators use them as their own applications via simple message exchanges. Only an invocation of a service can accomplish a generic task such as authorization, digital signature, and payment; however, a complex task often requires numerous interactions among services that collectively complete the task. Using processes is pivotal to services composition [16], in which a process model captures component services and their interaction logics (e.g., control flow, data flow, interdependencies) to meet the required functionalities. For a composite service to be effective and interoperable, every component service must be of high quality as well as functionality. However, solving the service composition problem aligned with discovery and selection is a non-linear engineering process due to many possible combinations with a number of different services discovered. The worse is that the problem is often stated in informal expressions.

The paper aims to formulate a multi-criteria composition problem in regular expressions. We first present a service allocation framework and list quality and functional attributes used in service allocation. Second, we discuss numerical methods to interpret functional attributes, especially, information compatibility. Third, we give formal definitions of service engineering functions. Last, a case example demonstrates the allocation problem.

The remainder of the paper is organized as follows: Next, Section 2 briefly states the web service allocation and lists service attributes. Section 3 presents formulations associated with the web service allocation problem. A case example and preliminary validation are given in Section 4, and Section 5 remarks the conclusion and future works.

2. Web Service Allocation

2.1. Service Allocation Procedure

The service allocation problem is to determine an optimal – both high functionality and quality – combination of services for a given composition model. Roughly stated, the allocation consists of discovering high functional services from a registry and then selecting a combination of high quality services from the discovered ones as shown in Figure 1. Service discovery is responsible for finding po-

tential services that undertake the functionalities required by the composition model, whereas service selection is for picking up the best set of services discovered (and their execution order) that maximizes the overall utility and quality of the resulting composite service. Suppose a simple example that a *TravelPlanner* system performs sub-tasks in the order of booking a flight, booking an accommodation, renting a car, and buying travel insurance. In this case, a composition model may state those tasks in that order. We first look up services providing the same functionalities as each task specified, generates various combinations of discovered services, and then chooses the best combination giving the highest quality.

In detail, the discovery activity explores the registry to find plausible services that perform desired functional operations. This activity receives a process-based composition model – referred to as a service-independent composition model (SICM) – as input and returns output in another composition model – referred to as a service-mapped composition model (SMCM). In particular, the input SICM only indicates all sub-tasks (in nodes) and their interaction logics (in directed arcs) necessary to accomplish a complex task. We assume that the SICM is complete, in that each sub-task has one or more corresponding services available, thereby requiring no further task decomposition. This also means that all the services available are assumed atomic. For each sub-task, the activity finds relevant services having designated functional attributes, and maps them to the corresponding task node. After this process, the SICM becomes an SMCM, each node of which is aligned with a set of alternative services discovered. This process is undertaken on **MatchMaker** that compares the functional attributes such as service names and I/O data definitions.

An SMCM can represent various execution plans that are feasible combinations of different services in different execution orders. Each execution plan provides the same functionalities, but may give a different overall quality assessment. In addition, each plan needs to support seamless data exchanges between component services. A directed acyclic graph (DAG) is used to represent an execution plan, in which a node specifies a service and an arc indicates execution precedence and information exchange between two services [16]. The plan allows concurrent executions (i.e., AND relations) if possible, but not alternative executions (i.e., OR relations). From an SCMC, **Scheduler** generates all the feasible execution plans.

Last, **Optimizer** evaluates those execution plans based on their quality assessments, QoS, and selects the best one as the final composite service. Selecting the final execution plan can be formulated as a multi-objective optimization problem to trade off between multiple quality attributes. In addition, since the evaluation uses averaged QoS in simulations and/or historical data, the final plan may not be the optimal one in a run-time. Therefore, Optimizer either should evaluate them in a run-time, or needs to prepare alternative plan(s) and/or to specify alternative service(s) for each task for run-time recovery.

2.2. Service Attributes

Service attributes include functional, non-functional, operational, and architectural requirements. We focus on functional attributes and quality attributes in this paper. First, the quality attributes, also called quality of service (QoS), represent non-functional properties of services. They are critical to select an optimal set of services among services providing the same functionalities, given resource constraints such as total operation cost and lead time. Typical quality attributes include **operation cost** to invoke a service, **performance** in terms of latency and throughput, **availability** in probability whether the service is ready for an immediate use, **accessibility** indicating the capability of serving requests, **security** in terms of confidentiality, authenticity, and integrity, **interoperability** between services, **reliability** of both message and service, and so forth [8] [10] [11] [16].

Second, the functional attributes represent the properties associated with a service's functionalities. Although a service is transparent to any users, they cannot recognize how the service exactly implements intended functionalities. In other words, "the only part of a service that is visible to the outside world is what is expressed via the service's description and formal contact, but the underlying process logics are typically invisible to service requesters [11]". Therefore, service discovery has to rely completely on explicit descriptions about a service, for example, its capability, its input and output. Such descriptions are available in WSDL (Web Service Description Language) [1], UDDI (Universal Description, Discovery, and Integration) [2], and OWL-S (Web Ontology Language for Services) [9]. Functional attributes include **service classification** such as the business domains to which the service belongs, **service name** that is used as a unique identifier and provides a high-level description of the service's functionalities, **operation name** for specific operations in a service, **data definitions** of input and output messages in XML Schema, and **annotation** for auxiliary descriptions about the service.

2.3. Interpreting Functional Attributes

Measuring quality of a service numerically may be straightforward by aggregating simulated data and/or historical operation data or more complicated when services have some interdependency. This is an issue of research not considered in this paper. On the other hand, matching functional attributes is different. The match degree can be quantified by means of semantic similarity of each func-

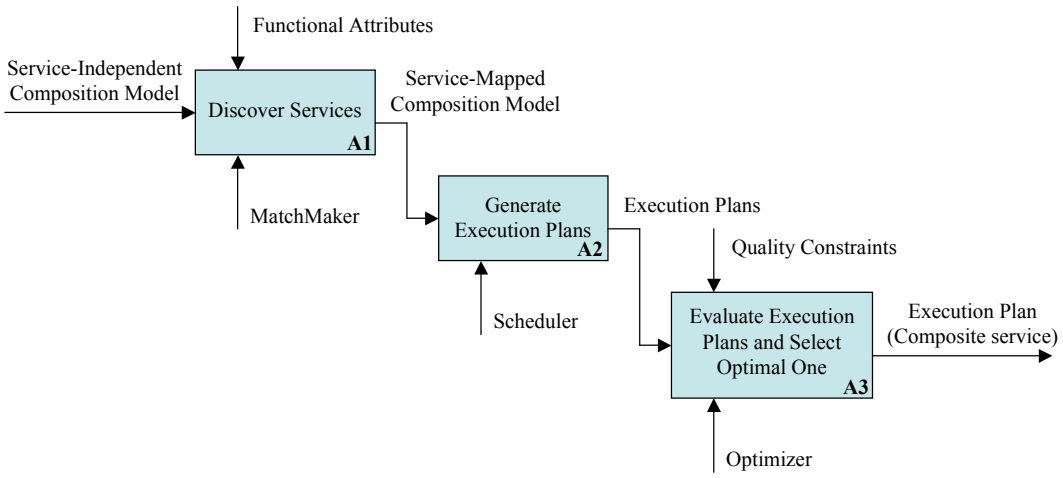

Figure 1. Allocating and optimizing web service composition

tional attribute. The functional attributes are classified into atomic labels (e.g., service category, service name), structured XML data (i.e., input and output definition), and textual data (i.e., plain annotation). For each of them, we need to approach in different ways to exploit its semantics.

First, the term similarity is used to quantify the commonality between atomic labels using purely lexical information. Term similarity measures include lexical form-based ones (e.g., prefix, suffix, n-gram, term edit distance) and semantics-based ones (e.g., word sense, synonym, information content) [4]. Generally, the semantics-based measures provide more accurate predictions, but require well-compiled lexical knowledge resources (e.g., WordNet).

Second, the input and output definitions, usually XML schemas, are expressed and structured in a tree, so that the best way to exploit such tree structures is to measure a tree similarity. A tree similarity counts the commonality in parent-to-child and right-to-left orders of nodes as well as in individual nodes themselves. For this reason, a tree similarity is more conservative than a term similarity. Examples include node/edge/path matching, inclusive path matching, tree edit distance, and kernel-based one [4] [5].

Last, for plain annotations, we can adopt techniques used in text mining. A typical one is the bag-of-words approach, particularly, including a vector space model (VSM) [13] and latent semantic analysis/indexing (LSA/LSI) [7]. The final text similarity is often computed as the cosine of angle defined by two vector representations.

2.4. Information Compatibility

We introduce an important analysis, namely **information compatibility**, in order to build a seamless and interoperable composite service. A number of services may exist that offer the same functionalities with slightly different data interfaces. That implies an inappropriate selection of component services may prevent a composite service from seamless data exchanges among its component services. The information compatibility analysis ensures correct data transfers from a service to succeeding services by mapping each information item in source data to corresponding one(s) in destination data.

Take a simple service connection as shown in Figure 2, in which the mapping tool merges, fragments, and transforms source data into destination data consumable by Service B. First, the compatibility analysis identifies the relation from sources to a destination, which is beyond the degree of similarity between them. For seamless connection, the sources must be sufficient for the destination. In other words, it is desirable that either a source or several sources collectively are identical to the destination (i.e., exact or full match), or the sources fully cover the destination (i.e., general match). At least, the sources should have overlaps with the destination (i.e., partial match). Second, the match must incorporate the information essentiality. For example, a mandatory information item in the destination must be available in the sources. Otherwise, a run-time transaction may fail. The information compatibility is measured as the portion of matched information items in the sources over the destination. In addition, we define **information discontinuity** as the opposite to information compatibility. Having large discontinuity means that more external inputs and/or information transformations are required.

3. Formulating Mulit-Criteria Composition

This Section provides formal statements necessary to define the service composition problem. The paper limit keeps us from providing in-depth descriptions of the statements and necessary equations.

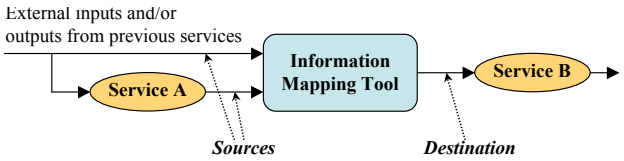

Figure 2. Service connection via information mapping

3.1. Service Composition and Invocation

A composite service is an ordered collection of services, and each service receives a message and returns a resulting message. This resembles a state machine, in that it transits from a state to another as if a service transforms an input message to an output message. Therefore, using the notations of the state machine, we formulate the service composition problem as a quintuple $C = (D, D_{in}, D_{out}, \Sigma, \delta)$, where D is a finite set of messages/data, $D_{in} \in D$ denotes an initial input, $D_{out} \subset D$ is a finite set of the final outputs, Σ is a finite set of atomic services (or operations), and $\delta : D \times \Sigma \rightarrow D$ is a data transition function, i.e., an operation trigger to transform an input to an output [4]. In particular, the top-level composite service produces D_{out} from D_{in}. An extension to this model is service invocation, defined as an octuple $I = (D, D_{in}, D_{out}, \Sigma, \delta, A, P, \gamma)$, where A is a set of operation actions that invoke services, P is a set of preconditions for invocation, and $\gamma : D \times \Sigma \rightarrow A$ is an operation-action transition function. Note that the definitions are not complete, but tentative to explain the underpinning ideas.

3.2. Service Discovery: A1

Service discovery finds potential services that not only meet functional requirements specified in a service query, but also are compatible with other connected services invoked already and/or to be invoked. A successive discovery and selection procedure that incorporates the functional attributes is presented as follows:

1. **Service Classification.** For a service query in an SICM node, look up all the services belonging to service categories – both explicit categories specified in the query and associated categories with the explicit ones. This look-up process compares the query with *tModel*'s in the UDDI registry.

2. **Service/Operation Name.** Sort services out that have names semantically equivalent to the name in the query. WSDL annotations, if available, are also used here, and human intervention may be required. The result is an SMCM.

3. **I/O Compatibility.** Identify service(s) whose input and output definitions are highly compatible with those of connected services as well as similar to those specified in the service query. This step may be iterative due to the intricate data transfers among services. This produces a set of execution plans.

4. **Quality Attributes.** Finally, optimize and select the best configuration for the composition model using the quality attributes.

3.3. Execution Plan Generation: A2

As described, an execution plan is a feasible composition of services that completes the task specified in the composition model. The feasibility is granted just by serializing the composition model and assigning one of any discovered services to each node. However, this can produce a number of less useful plans having large information discontinuity, thereby resulting in lack of interoperability between component services. Hence, an acceptable execution plan should be both functionally feasible and information compatible. Therefore, we can formulate this as: **Generate execution plans such that, for every source-destination pair (D_s, D_d), its information discontinuity is no more than a tolerance ϵ and the plan is able to accomplish the given task**. An execution plan needs additional inputs from external applications if information discontinuity exists.

3.4. Service Selection: A3

For functionally guaranteed execution plans, service selection is to evaluate them and select the best one in terms of QoS such as performance and availability. The original selection problem can be stated as: **Select the best execution plan such that it maximizes overall quality and utility (QoS) within resource constraints while minimizing total information discontinuity**. The problem is very complicated because it must optimize both of the two different objectives. However, we already obtained acceptable execution plans whose total information discontinuity is no more than Υ ($= \sum \epsilon$)[1] in the previous step. Accordingly, the problem approximately becomes: **Select the best execution plan from candidates such that it maximizes overall quality and utility within resource constraints**. It is noted that this approximate problem is still a multi-objective optimization problem that must simultaneously maximize multiple quality attributes.

[1] By Liebig's Law of Minimum, this may be re-stated as 'maximum information discontinuity between any pair of services is no more than ϵ'.

4. Preliminary Validation

4.1. Illustrative Example

Take the *TravelPlanner* application for a case example. As shown in Figure 3, the *TravelPlanner* application, by passing messages, orchestrates all the other services *AirlineBooking*, *HotelReservation*, and *CarRental*, either concurrently or sequentially, to complete a travel itinerary. The messages exchanged include **TI** (travel information), **TIX** (travel information provided to a service), **AS** (airline schedule), **HRC** (hotel reservation confirmation), etc.

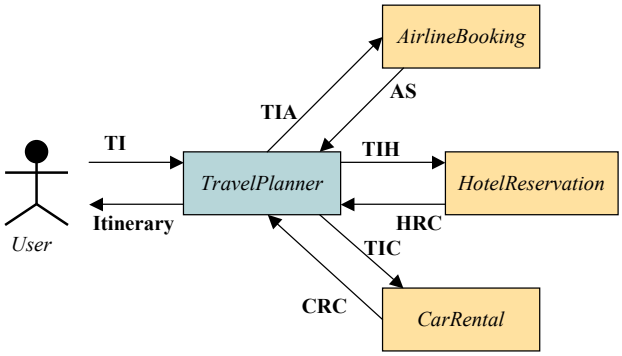

Figure 3. A service composition example

First, we can depict an SICM in a state diagram, particularly a Petri net [12][2], as shown in Figure 4, in which a transition node represents a service (or a corresponding operation in the service) and a place node represents an I/O message going to or coming from a service. The composition model needs to have functional descriptions about each node for discovery. Particularly, a data node contains corresponding data definitions in XML schema, whereas a service node has descriptions about the service such as service classification, service name, and annotation.

Second, several services may correspond with a service node. For example, reserving an accommodation may have different services – *HotelReservation* and *AccommodationBooking*, which use either the same I/O definitions or different I/O definitions. Third, suppose that **TI**, the external input, has only date fields for specifying dates to depart and return. If the *HotelReservation* service requires only those dates, whereas the *AccommodationBooking* service requires both dates and times, then we prefer to use *HotelReservation*. Or, if the *AirlineBooking* service returns flight times as output and if the *CarRental* service requires the times to rent and return, then invoking *CarRental* after *AirlineBooking* is more seamless than the opposite. Last, the service

[2]See [3] for basic Petri net notations for modeling a service composition model.

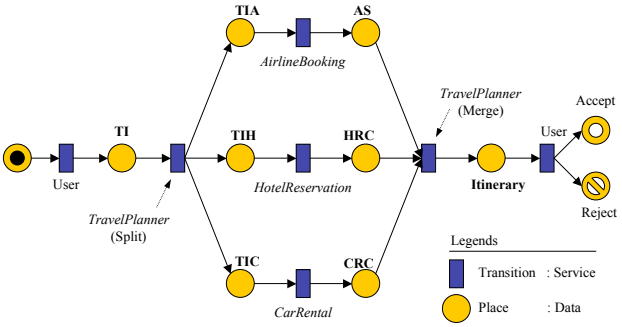

Figure 4. A Petri net model of service composition

selection is straightforward to evaluate alternative execution plans and pick the best one up. For example, if the allowable cost to invoke a service to reserve a room is up to $2 and if *HotelReservation* and *AccommodationBooking* require $1 and $2, respectively, then the final solution prefers to include *HotelReservation*. However, the selection problem becomes more complex as more quality attributes are considered, for example, their respective availability are 0.6 and 0.9.

4.2. Service Discovery Simulation

We conduct an experiment using a collection of web services from XMethods[3] [6] [14]. It has a total of 38 services from five categories: weather information finder (6), currency rate converter (7), DNA information searcher (5), SMS sender (10), and ZIP code finder (10). We use the operation name, input message type, and output message type as functional attributes. In particular, Wu and Palmer's algorithm [15] is used to compute term similarity between operation names, while the kernel-based measure [5] is used as a tree similarity measure for input and output message types. An average and a weighted-average are also used to aggregate those individual measures. Fig. 5 depicts a pairwise proximity matrix for the operation name. The matrix shows us that web services in the same category tend to be more similar (dark color), whereas services between different categories are less.

Using the PAM (Partitioning Around Medoids) algorithm, we cluster those web services. In short, ratios of correct classifications are approximately from 0.74, 0.84, 0.76, 0.89, and 0.89 for operation name, input type, output type, average, and weighted average, respectively. Although each functional attribute alone discriminates those services well, the integrated measures outperform individuals.

[3]http://www.xmethods.com

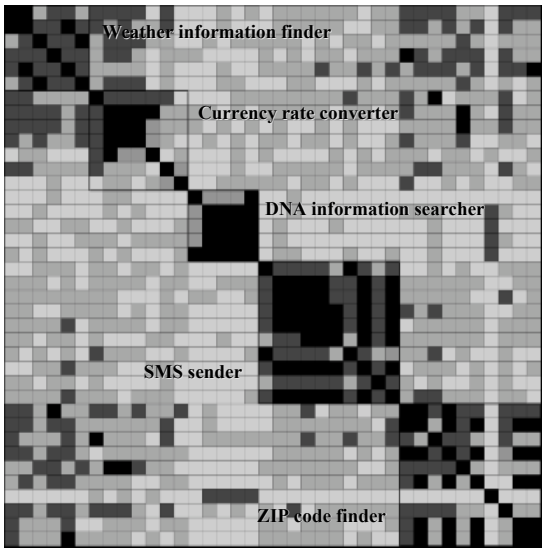

Figure 5. A demonstrative proximity matrix of the operation name: Deeper color indicates more similar.

5. Conclusion

The service-oriented approach (SOA) has undeniably been the popular choice these days to develop, integrate, and deploy enterprise applications. To share common understandings of SOA across researchers is important. For that purpose, the paper addressed the multi-criteria service allocation problem, which subordinates the problems of service composition, discovery, and selection. In particular, services are selected based on functional attributes as well as quality attributes. The paper provided provisional definitions to state and solve those problems. Even though the paper is not the first one dealing with such problems, it makes quite an impact on their formalization. The definitions are tentative, yet solid, and not in depth; therefore, we need to consolidate them further and apply to real integration scenarios. The formal techniques need to be represented in a machine-interpretable manner and geared with semantic descriptions of service attributes for automated engineering.

Disclaimer

Certain commercial software products are identified in this paper. These products were used only for demonstration purposes. This use does not imply approval or endorsement by NIST, nor does it imply that these products are necessarily the best available for the purpose.

References

[1] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Service Description Language (WSDL) 1.1*, Mar. 2001.

[2] L. Clement, A. Hately, C. von Riegen, and T. Rogers. *Universal Description, Discovery, and Integration (UDDI) 3.0.2*, Oct. 2004.

[3] H. Gou, B. Huang, W. Liu, S. Ren, and Y. Li. Petri net-based business process modeling for virtual enterprises. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (SMC2000)*, pages 3183–3188, Oct. 2000.

[4] B. Jeong. *Machine Learning-based Semantic Similarity Measures to Assist Discovery and Reuse of Data Exchange XML Schemas*. PhD thesis, Department of Industrial and Management Engineering, Pohang University of Science and Technology, June 2006.

[5] B. Jeong, D. Lee, H. Cho, and B. Kulvatunyou. A kernel method for measuring structural similarity between xml documents. In *Proceedings of the 20th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems (IEA/AIE-2007)*, June 2007.

[6] N. Kokash. A comparison of web service interface similarity measures. In *Proceedings of the European Starting AI Researcher Symposium (STAIRS)*, pages 220–231, Aug. 2006.

[7] T. Landauer, P. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse Processes*, 25:259–284, 1998.

[8] A. Mani and A. Nagarajan. Understanding quality of service for web services. Technical report, IBM, Jan. 2002.

[9] D. Martin. *OWL-S: Semantic Markup for Web Services*, Oct. 2003.

[10] D. Menasce. QoS issues in web services. *IEEE Internet Computing*, pages 72–75, Nov. 2002.

[11] L. O'Brien, L. Bass, and P. Merson. Quality attributes and service-oriented architectures. Technical Report CMU/SEI-2005-TN-014, Carnegie Mellon University, Sept. 2005.

[12] J. Peterson. Petri nets. *ACM Computing Survey*, 9(3):223–252, 1977.

[13] G. Salton, A. Wong, and C. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[14] J. Wu and Z. Wu. Similarity-based web service matchmaking. In *Proceedings of the 2005 IEEE International Conference on Service Computing (SCC'05)*, pages 287–294, July 2005.

[15] Z. Wu and M. Palmer. Verb semantics and lexical selection. In *Proceedings of the 32nd Annual Meeting of the Associations for Computational Linguistics*, pages 133–138, 1994.

[16] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng. Quality driven web services composition. In *Proceedings of WWW2003*, pages 411–421, May 2003.