Algebraic modeling and verification of Web service composition

Gopal N. Rai\textsuperscript{a,b}, G. R. Gangadharan\textsuperscript{a,*}, Vineet Padmanabhan\textsuperscript{b}

\textsuperscript{a}IDRBT, Masab Tank, Hyderabad-500057, India
\textsuperscript{b}SCIS, University of Hyderabad, Gachibowli, Hyderabad-500046, India

Abstract

In order to provide a rigorous and sound foundation for formal reasoning about Web services, algebraic modeling is one of the important techniques used as is witnessed from the Web service literature. However, the algebraic modeling approach for Web services (Web service algebra) is still in its infancy. To further facilitate the algebraic modeling of Web services, in this paper, we propose a composition algebra based on the notion of recursive composition. The proposed algebra is fully capable to verify the presence of behavioral equivalences and deadlock conditions in a Web service composition scenario. The main motivation for proposing Web service composition algebra is to capture the recursive nature of composition which cannot be done using traditional approaches like model checking and Petri net.

© 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Keywords: Web service algebra; Web service composition; formal methods; modeling

1. Introduction

Two distinguishing characteristics of Web service composition are that of interaction through messages and recursive composition among Web services. Since Web services are deployed heterogeneously and independently, composition of these Web services may lead to subtle concurrency related issues. Modeling and realization of the composition is not easy with the help of Kripke\textsuperscript{1} or Petri net\textsuperscript{2} modeling techniques if the number of services involved in the composition is high. Since a Web service composition scenario could be seen as a model of communicating systems and inherits the concept of concurrency, process algebras\textsuperscript{3,4,5} could be a good choice for modeling it as has been shown to a certain extent in\textsuperscript{6,2,7}. Nevertheless, Web service composition pertains to a different class of communication behaviors that demand its own constructs and operators. In order to satisfy such requirements, we propose a Web service algebra that can model and verify Web services in a much more efficient manner. Though researchers have proposed several algebras focused on Web services\textsuperscript{8,9,10}, these do not give emphasis on recursive nature of service composition and also do not explore on the verification aspect of composition. Therefore, we emphasize that

\textsuperscript{*} Corresponding author. Tel.: +91-40-2329-4131; fax: +91-40-2353-5157.

E-mail address: geeyaar@gmail.com
a Web services algebra should have its own basic computational model on recursive composition where interaction must be one of the basic activities (primitive notions) of the model.

In this paper, we propose a simple yet powerful recursive composition based algebra for modeling and verifying Web service compositions. The proposed algebra consists of a set of Web services and three algebraic operators: successor, composition, and recursive composition. The proposed algebra provides a sound foundation for verification and reasoning. In order to verify the Web service composition, we form a set of canonical sets out of Web services set. The process of deriving canonical sets employs the operators proposed in the algebra. The results inferred by canonical sets are used to verify behavioral equivalence and deadlock among Web services.

The remainder of this article is structured as follows. Description of our proposed operators and algebraic structure with the concept of canonical sets are available in section 2. Section 3 presents the relevant proposals of Web service algebra found in literature and their expressiveness. Section 4 provides concluding remarks and gives future directions.

2. Proposed Web service algebra

Let \( \mathcal{W} = \{w_1, w_2, w_3, \ldots, w_m, \epsilon\} \) be a finite set of available Web services, where \( \epsilon \) represents an empty Web service. An empty Web service does not invoke any service or perform any activity. Throughout the paper, we consider \( \mathcal{W} \) in the same meaning unless stated otherwise. We define a Web service \( w_i \in \mathcal{W} \) as follows:

\[ \text{Definition 2.1 (Web service). A Web service } w_i \in \mathcal{W} \text{ is 3-tuple } (I, O, R_l), \text{ where } I = \{I_1, \ldots, I_p\}, \ p \in \mathbb{N} \text{ is a finite set of input messages, that } w_i \text{ accepts. } O = \{O_1, \ldots, O_q\}, \ q \in \mathbb{N} \text{ is a finite set of output messages, that } w_i \text{ produces. } R_l \text{ is a relation that maps an input message from } I \text{ to output messages in } O \ (R_l \subseteq I \times O). w_i, I, w_i.O, \text{ and } w_i.R_l \text{ are referred as the set of input messages, the set of output messages, and the relation from } w_i.I \text{ to } w_i.O \text{ in } w_i. \]

For a Web service, the set of input messages, the set of output messages, and relation from input message set to output message set are static and available in the respective Web Service Description Language (WSDL) file.

2.1. Operators (Successor, composition, and recursive composition)

\[ \text{Definition 2.2 (Absolute successor). Let } ' \succ ' \text{ be a symbol to represent the absolute successor operator. } \succ \text{ maps an element of the } \mathcal{W} \text{ to an element of the power set of the set } \mathcal{W} (\succ: \mathcal{W} \rightarrow 2^\mathcal{W}). \text{ Given a web service } w_i \in \mathcal{W}, S \subset \mathcal{W} \text{ is absolute successor of } w_i \text{ if and only if } \forall w_j \in S : w_j.O \cap w_i.I \neq \emptyset. \]

In other words, an absolute successor operator is an unary operator that provides services directly invokable by a service (we call them as successor services). Consider that a service \( w_i \in \mathcal{W} \) invokes a service \( w_j \in \mathcal{W} \). Then \( w_j \in (\succ w_i) \). If \( w_j \) is not known in advance, we write \( w_j = w_i + 1 \) unless stated otherwise. If the service \( w_i \) directly invokes a set of services \( \{w_1, \ldots, w_l\} \subset \mathcal{W} \) then \( w_j = \{w_1, \ldots, w_l\} \). If the service \( w_j \) does not invoke any service from the set \( \mathcal{W} \) then \( w_j = \epsilon \).

Composition of services (say \( n \in \mathbb{N} \), no. of services) is the aggregation of facilities provided by the \( n \) services as a single service. Sequential composition and parallel composition are the two typical behaviors of the composition process. Let '\' \( \oplus \) ' and ' \( \otimes \) ' be two symbols that represent the sequential composition and parallel composition respectively. We define sequential composition and parallel composition as follows.

\[ \text{Definition 2.3 (Sequential composition). Given two Web services } w_i, w_j \in \mathcal{W} : w_j \in (\succ w_i), \text{ sequential composition of } w_i \text{ and } w_j \text{ (represented as } w_i \oplus_s w_j) \text{ yields a composite service } w_k \in \mathcal{W} \text{ such that } \]
\[ (\forall m \in w_i.I : (w_i.Rl(m) = n) \land (n \in w_j.I) \rightarrow ((m \in w_k.I) \land (w_k.Rl(m) \subseteq w_j.Rl(m)))) \quad (1) \]

\[ \text{Definition 2.4 (Parallel composition). Given two Web services } w_i, w_j \in \mathcal{W} : w_j \notin (\succ w_i), \text{ parallel composition of } w_i \text{ and } w_j \text{ (represented as } w_i \oplus_p w_j) \text{ yields a composite service } w_k \in \mathcal{W} \text{ such that the input message set of } w_k \text{ is consolidation of the input message sets of } w_i \text{ and } w_j \text{ and the output message set of } w_k \text{ is consolidation of the output message sets of } w_i \text{ and } w_j. \]
\[ w_i \oplus_p w_j \triangleq \left[ w_k : (w_k.I = (w_i.I \cup w_j.I)) \land (w_k.O = (w_i.O \cup w_j.O)) \land (w_k.Rl = (w_i.Rl \cup w_j.Rl)) \right] \quad (2) \]
Let symbol ‘⊕’ be a common representation for both sequential composition operator and parallel composition operator (removing the suffixes s and p from ⊕s and ⊕p).

Let \( w_i, w_j \in \mathcal{W} \) be two Web services such that their composition \((w_i \oplus w_j)\) is possible. However, the resultant Web service \((w_k)\) for the composition does not exist in the set \( \mathcal{W} \). Then, \( w_j \oplus w_i \) itself represents a composite service that is able to participate in further composition processes as a single service. However, composition with empty service results as a service itself without any change \((w_i \oplus \epsilon = w_i)\).

**Definition 2.5** (Conditional successor). Conditional successor \((\succ_c)\) accepts input and produces output in the form of a 2-tuple \( \langle w_i, I_p \rangle \) where \( w_i \in \mathcal{W} \) and \( I_p \in w_i.I \). Given a tuple \( \langle w_i, I_p \rangle \), \( \langle w_j, I_r \rangle \) is a conditional successor of \( \langle w_i, I_p \rangle \) \((\succ_c \langle w_i, I_p \rangle)\) if and only if \( w_j \in (\succ w_i) \) and \( I_r \in w_i.Rl(I_p) \).

Conditional successor for a tuple with empty second field (input message is not specified) considers all input messages of the respective Web service and produces the output tuples accordingly. Restrictive successor operator \((\succ_r)\) is a conditional successor operator such that \( \text{Domain}(\succ_r) = \text{Domain}(\succ_c) \) and \( \text{Range}(\succ_r) \subseteq \text{Range}(\succ_c) \). We formally define restrictive successor operator as follows.

**Definition 2.6** (Restrictive successor). Let \( \langle w_i \oplus \cdots \oplus w_n, I_p \rangle \) and \( \langle w_s, I_r \rangle \) be two 2-tuple arguments (as considered in the definition of conditional successor), where \( I_p \in w_i.I, I_r \in w_s.I \), and \( w_i, w_s, w_k \in \mathcal{W} \) then \( \langle w_i, I_r \rangle \in (\succ_r \langle w_i \oplus \cdots \oplus w_n, I_p \rangle) \) if and only if \( \langle w_s, I_r \rangle \in (\succ_c \langle w_i \oplus \cdots \oplus w_n, I_p \rangle) \) and \( w_s \notin \{w_i, \ldots, w_n\} \).

Let ‘⇝’ be a symbol to represent recursive composition. To define recursive composition, we incorporate restrictive successor operator \((\succ_r)\) and composition operator (⊕) as supplementary operators (defined earlier in this section).

**Definition 2.7** (Recursive composition). Recursive composition for a given Web service \( w_i \in \mathcal{W} \) is defined by the following piecewise function

\[
\oplus w_i = \begin{cases} 
\epsilon & \text{if } w_i = \epsilon \\
\epsilon \cup \{ \oplus (\succ_r w_i) \} & \text{if } w_i \neq \epsilon \\
\{ w_i \} & \text{otherwise}
\end{cases}
\]

Recursive composition on \( w_i \) generates a directed tree with \( w_i \) as root. We call every path (from the root to the leaf) in the tree as a *trace*. Let \( w_i \) be a Web service then \( T_{w_i} \) represents a set which contains all the traces generated by applying the recursive composition on \( w_i \). We follow the concept of trace mainly while studying behavioral equivalence of services. Successor operator and recursive composition operator are distributive over a set of Web services. But they are not distributive over composition operator.

### 2.2. Canonical set using recursive composition

The term *canonical* is not an absolute one. It gives meaning to the word adjoining it. The use of the word canonical set varies from context to context in mathematics, logic, and algebra. We redefine the term canonical set in the context of our proposed algebra as follows.

**Definition 2.8** (Web service canonical set for a Web service \( w_i \)). Given the set \( \mathcal{W} \), a canonical set \( C_i \) for a Web service \( w_i \in \mathcal{W} \) with respect to the set \( \mathcal{W} \) is a subset of \( \mathcal{W} \) such that it consists all leaf nodes (other than the root node) of the tree generated from application of recursive composition operation on the service \( w_i \).

Let \( C_i \) be a canonical set for a Web service \( w_i \) and ‘⇝’ be a symbol to represent ‘leads to’. Then, \( \oplus w_i \leadsto C_i \). Even if a service does not invoke any service, empty canonical set exists. For instance, if \( \oplus w_i = \epsilon \) then \( \oplus w_i \leadsto C_i = \emptyset \).

The computation of canonical sets for all web services yields a partition set \( C \) of the set \( \mathcal{W} \). The partition set \( C \) consists in number of sets \( \mathbb{C} = (C_1, \ldots, C_m) \). \( C_i \) is the canonical set generated by \( w_i \) where \( \emptyset < i \leq m \).

Canonical sets play a major role in composition and verification process of Web services. Let \( S_{is}, S_{ig}, \) and \( S_{im} \) be the subsets of \( \mathcal{W} \) and be the sets of isolated, igniter, and terminator Web services respectively. Several logical interpretations based on the proposed algebra and canonical sets are deduced and discussed with their significance as follows:
1. An isolated Web service is one that cannot be invoked by other services as well as cannot invoke other services. Excluding isolated services out from the set of Web services is mandatory as their presence in the Web services set increases the computational overhead during composition. On the basis of recursive composition operator and canonical sets, isolated services can be recognized automatically as follows:

\[ w_i \in S_{is} \leftrightarrow \left( \exists w_j \in W : (\oplus w_i \leadsto C_i = \emptyset) \land (\oplus w_j \leadsto C_j) \land (w_i \in C_j) \right) \]  (4)

2. A strict igniter service is one that cannot be invoked by other services but can invoke other services.

\[ w_i \in S_{ig} \leftrightarrow \left( \exists w_j \in W : (\oplus w_i \leadsto C_i = \emptyset) \land (\oplus w_j \leadsto C_j) \land (w_i \in C_j) \right) \]  (5)

A strict terminator service is one which does not invoke any service but be invoked by other service.

\[ w_i \in S_{tm} \leftrightarrow \left( \exists w_j \in W : (\oplus w_i \leadsto C_i = \emptyset) \land (\oplus w_j \leadsto C_j) \land (w_i \in C_j) \right) \]  (6)

Classifying igniter services and terminator services into separate sets reduces the search space while verifying composition. By leveraging the condition \( \exists w_j \in W \left[ w_i \in C_j \right] \) and \( \exists w_j \in W \left[ w_i \in C_j \right] \), non-strict igniter and terminator services can be computed.

3. Behavioral equivalences serve as the conceptual basis for verifying that the behavior of two Web services can be considered to be the same. Given a Web service \( w_i \in W \), the fulfillment of the following condition reflects that \( T_{w_j} \) consists all the behavior represented by \( T_{w_i} \) but converse is not true.

\[ \exists w_j \in W : (\oplus w_i \leadsto C_i) \land (\oplus w_j \leadsto C_j) \land (C_i \subseteq C_j) \]  (7)

The following condition implies that behaviors represented by \( T_{w_i} \) are partially equivalent to behaviors represented by \( T_{w_j} \) and vice versa.

\[ \exists w_j \in W : (\oplus w_i \leadsto C_i) \land (\oplus w_j \leadsto C_j) \land (C_i \cap C_j \neq \emptyset) \]  (8)

The following condition implies that the complete behavioral equivalence exists between \( T_{w_i} \) and \( T_{w_j} \).

\[ \exists w_j \in W : ((\oplus w_i \leadsto C_i) \land (w_j \in C_i)) \land ((\oplus w_j \leadsto C_j) \land (w_i \in C_j)) \land (C_i \setminus w_j = C_j \setminus w_i) \]  (9)

This condition reflects that for a trace \( T_{p} \in T_{w} \) there exists \( T_{q} \in T_{w} \) such that \( T_{p} \) can be substituted with \( T_{q} \) and vice versa.

4. Deadlock conditions can be detected with the help of canonical sets. Given two Web services \( w_i \) and \( w_j \), the fulfillment of the following condition infers that the traces generated by \( w_i (T_{w_i}) \) and \( w_j (T_{w_j}) \) may lead to deadlock condition.

\[ ((\oplus w_i \leadsto C_i) \land (w_j \in C_i)) \land ((\oplus w_j \leadsto C_j) \land (w_i \in C_j)) \]  (10)

3. Related Work

Hamadi and Benatallah\(^\text{11}\) propose a Web service algebra based on the Petri net model for representation of services and interpretation of syntaxes of grammar. The proposed Web service algebra\(^\text{11}\) consists of an empty Web service, set of basic Web services, and eight different operators. Although this work\(^\text{11}\) is considered as one of the novel approaches in Web service algebra, it is difficult to realize each service as a stand alone Petri net model, considering the seamless proliferation of Web services nowadays. We addressed all these above mentioned issues in our proposal. Hashemian and Mavaddat\(^\text{8}\) present a composition algebra as an alternative way of representing interface automata. This composition algebra captures the behavior of services and their composition. It employs two basic operators: sequence, and choice; two parallel operators: parallel, and synchronization and some more operators like hiding operator borrowed from process algebra. Authors explicitly mention that there is no direct or indirect recursion allowed in process composition\(^\text{8}\). In our work, we described the importance of recursion operator and proposed algebra focused on recursive composition.
Hoefner and Lautenbacher$^{10}$ present an algebraic structure of Web services which assist users in Web service composition and formal description of their services. Using relation algebra, tests and iterations offer the possibility of an automatic composition of Web services based on a specified goal. However, the syntax, and semantics used in$^{10}$ do not seem intuitive for modeling and verification process.

Hu et al.$^{9}$ propose service net algebra (SNA) based on logic Petri net model. They emphasize the reuse of a service process instead of reuse of a composite Web service. A biggest drawback as mentioned by authors in$^{9}$ is that the Petri net is not suitable to model the complicated Web service composition. Our focus is on to verify composition and behavioral equivalence whereas$^{9}$ focuses on reuse of service process.

Yu and Bouguettaya$^{12}$ propose a Web service query algebra based on a formal service model that provides a high level abstraction of Web service across an application domain, aiming to describe a solution to service query. In our proposed algebra, we aim for service composition and verification.

The aim and scope of process algebra are slightly different from the Web service algebra. However, several researchers applies process algebra in Web services$^{6,13,7}$. In general, process algebras are good in detecting behavioral equivalence or bisimulation study. Even though process algebra treats the movement of a message across the chain of services very nicely, it differs from our proposed algebra in its treatment of mobility. A link between two processes is mobile in process algebra whereas a link is stable in the case of modeling communication among Web services.

4. Conclusion and Future Work

In this paper, we present a recursive composition based algebra for composition and verification of Web services. On applying over the set of Web services the recursive composition operator yields a set of canonical sets. To the best of our knowledge, the proposal and application of recursive composition algebra and canonical sets in service composition and verification is completely new. Logical interpretations deduced from canonical sets make verification process very easy and consumes less computation time and memory space. As the goal of a modeling technique is to express an intended class of scenarios efficiently and completely, our proposed algebraic model works as a super model for other algebraic models for Web services. It does not exclude expressiveness of the other models. The proposed recursive composition algebra is not only limited to modeling to Web service composition but also applicable to various other logical and mathematical modeling disciplines. In our future work, we will enhance the proposed algebra with the provision of dynamic inclusion and exclusion of services in the set of Web services. Furthermore, we will enrich the proposed algebra with the matching process of Web services.

References