SIMILARITY MEASURES FOR WEB SERVICE COMPOSITION MODELS

Maricela Bravo

Systems Department, Autonomous Metropolitan University, Mexico City, Mexico

ABSTRACT

A Web service composition is an interconnected set of multiple specialized Web service operations, which complement each other to offer an improved tool capable of solving more complex problems. Manual design and implementation of Web service compositions are among the most difficult and error prone tasks. To face this complexity and to reduce errors at design time, the developer can alternatively search and reuse existing compositions that have solved similar problems. Thus the problem of designing and implementing Web service compositions can be reduced to the problem of finding and selecting the composition closest to an initial specification. To achieve this goal, there is the need to define and use similarity measures to determine how close is a given composition with respect to any given specification. Comparison of Web service compositions can be done using two possible sources: composition designs (models), and execution logs of compositions. In particular, in this paper a set of similarity measures are described for Web service composition models. The main objective is to measure and assess the degree of closeness between two given compositions of Web services regardless of their modelling language.

KEYWORDS

Similarity measures; Web service compositions; State similarity

1. INTRODUCTION

A Web service composition is an interconnected set of multiple specialized Web service operations, which complement each other to offer an improved tool capable of solving more complex problems that go beyond each individual service capability. Manually designing and implementing Web service compositions is among the most difficult and error prone tasks that any application developer may face. Based on a given initial specification of a complex problem, the common steps that the composition developer must follow are: identify specific sub-problems derived from the complex problem; look for software components (in the form of Web services) that can solve each sub-problem; design the information flow and execution flow for the overall Web service composition; and finally, collect all responses and integrate them into a global composed response to the client.

In order to face this complexity and to reduce the design time for Web service compositions, the composition developer could alternatively search and reuse existing compositions that have solved similar problems. Such Web service compositions can be found in repositories of Web service descriptions which publish simple and compound services. Thus the problem of designing and implementing Web service compositions can be reduced to the problem of finding and selecting the composition closest to the initial specification.

DOI : 10.5121/ijwsc.2014.5101

To achieve this goal, there is the need to use similarity measures to determine how close is a given composition with respect to the initial specification. Similarity measures applied to Web services are not a new subject, as they have been studied and addressed long ago, this is because application developers have faced many times, the problem of searching and selecting simple Web services to meet their specific needs. However, to date little progress has been made in relation to the construction of public repositories of Web service compositions that provide proven solutions to common problem specifications. Also, little progress has been published in relation to the complex tasks of searching, selecting and matchmaking composed Web services.
Regarding the comparison of Web service compositions, this task can be done using two possible resources: the composition designs (models), and the execution logs of compositions. Both source options pose different challenges and difficulties mainly because of the representation format and techniques required to extract information. In case of comparing composition designs or models, Web service compositions could have been modelled, described or implemented in some of the following languages: BPEL4WS , BPMN , EPC , YAWL , WS-CDL among others. In case of comparing execution logs of Web service compositions, it is necessary to mine server logs in order to extract and analyze the sequences of messages issued during execution at the hosting Web server.

In particular, in this paper the set of similarity measures are proposed for the comparison of Web service composition models. In order to provide a solution compatible with majority of composition languages, no specific modelling language is required. Instead, a more general formalism representation is used. The main objective is to measure and assess the degree of closeness between two given compositions of Web services regardless of their modelling or representation language. Reported works in the literature describe a wide range of similarity measures applicable to Web services. Methods for comparing Web services are usually based on syntax or semantic approaches which frequently fail to identify similar service operations based on their observable behaviour [1]. However, among all these similarity measures, none has addressed aspects of the expected behaviour analysed from the design of the composition.
The rest of the paper is organized as follows. In the next section related works are presented. In Section 3, the similarity measures for Web service compositions are introduced. In Section 4 experimental results are described. In Section 5, evaluation of results is presented. Conclusions close the paper in Section 6.

2. RELATED WORK

In this section, an overview of relevant similarity measures approaches is described, these approaches range from pure syntactical, semantic to behavioural.

2.1. Syntactic Measures

Syntactic similarity measures are those based on lexico-syntactical comparisons, such as a string to string comparison. Whereas structural similarity measures are those that exploit signature or interface information of the service, such as: input parameters, output parameters, operation names and descriptions, and service name and descriptions [14]. Some distance measures to compare strings have been applied to Web services, for instance the Hamming distance used in information theory [2]; or the Levenshtein [3] distance between two strings. The UDDI Registry By Example (URBE) for Web service retrieval uses the evaluation of similarity between Web service interfaces [4]. The algorithm combines the analysis of the interfaces structures and the analysis of the terms used inside them. URBE is useful to find a Web service suitable to replace an existing one that fails. Semantic Annotation for WSDL (SAWSDL) is adopted as a language to annotate a WSDL description. Dong et al. [5] provide a syntactic-structural approach for supporting Web service similarity search and clustering; service name and descriptions of text, operation, and input/output are considered over agglomerative algorithm to discover similar operations and input/output parameters.

2.2. Semantic Measures

Semantic similarity measures use dictionaries or semantic references such as ontologies to construct representations of the meanings. Bruno et.al [6] describe a semantic-based approach for automated classification and annotation of Web service description files (WSDL). Their approach relies on Support Vector Machines (SVM) and Information Retrieval Vector Spaces for service classification. As an outcome they build concept lattices representing service domains. Stroulia and Wang [7] developed and evaluated three methods to assess the similarity between two WSDL specifications. The first method is based on the Vector-space model information retrieval and WordNet considering words at the lexical level only. The second method represents an extension of the signature-matching method for component retrieval; this method is a structural-based approach. And the third method combines the structure-based matching with a semantic approach, by using WordNet to calculate the semantic distances between each pair of compared elements in the WSDL specifications. The semantic Web has influenced many works by providing logic-based mechanisms to describe, annotate and discover Web services. Within this context, McIlarith, Cao Son & Zeng [8] proposed one of the first initiatives to markup Web services based on DAML (ontology language), which started the important research area of “Semantic Web Services”. The term Semantic Web Services is related to the set of technologies to design or implement ontologies as a mechanism to enhance or annotate semantically service descriptions, for instance OWL-S , WSMO , and SAWSDL. To use these technologies there is the general assumption that the annotation or ontology representation is done previously or during Web service deployment. An attempt to characterize and improve Web service search and discovery using a semantic approach called "conceptual indexation" is presented in [17]. Conceptual indexation method is based on OWL-S service descriptions to exploit *hasInput* and *hasOutput* elements to index Web services and requests.

2.3. Behavioural Measures

From the perspective of behaviour analysis applied to Web services interaction, there are some reported works in the literature. A functional quality of service approach to discover and compose interoperable Web services is described in [1]. They consider as functional attributes the service category, the service name, the operation name, the input and output messages and the annotation of the service. A tree similarity based on structure matching of XML schema documents, is provided. Grigori et.al [9] address the problem of service behavioural matching by implementing a graph matching algorithm which is based on the edit distance similarity measure. They apply their behavioural approach to service matchmaking, where a user service request is represented as a process graph, aiming at comparing against a set of published models graphs in a library. Similarly, Dijkman et.al [10] compare four graph matching algorithms to discover business process similarity. In [11] authors describe Match and Merge operators for Statecharts, their Match operator is based on typographic and linguistic similarities between the vocabularies of the different models producing a corresponding relation between the states of compared models; their Merge operator produces a merge that: preserves the behavioural properties of the models, respects the hierarchical structures, and distinguishes between shared and non-shared behaviours of compared models.

Despite the existence of numerous proposals to measure the similarity of Web services, it has not been possible to achieve the desired level of automation for the selection and discovery of Web services. Measurement approaches based on syntax and structure provide information related to the communication interface of the service, but say nothing about the functional and contextual information. On the other hand, semantic approaches strongly depend on human intervention, because there is always the requirement to manually annotate or enhance semantically Web services, providing their IOPE (input output preconditions and effects) annotations.

Approaches that analyze the behaviour of interactive systems applied in the modelling and comparison of Web service compositions are very suitable. For instance, the current trend for computing the context associated to a given situation described by Singh [13] for the Pragmatic Web represents a close antecedent to the work reported in this paper. Another important issue towards the automated measurement, selection and composition of Web services is understanding the "composition life cycle" presented by [18]. However, until now the analysis that combines measurements of structural and semantic similarity with these behavioural approaches have not been proved for composition design models.

3. SIMILARITY MEASURES FOR WEB SERVICES COMPOSITIONS

A Web service composition is the interconnection of multiple specialized Web service operations, which complement each other to offer an improved tool capable of solving more complex problems. Normally, the invocation of a service operation produces an output which may be part of the input of another service operation. A common form of interaction between single services inside a composed service is by means of an intermediary, which is a program responsible for creating required input objects, invoking service operations and receiving the output results from service operations. Is in the intermediary program, where data transformations are executed and where a general response is created as a result of the composed service. The automatic generation of the response is done by exchanging input and output messages between the different interconnected operations that form a Web service composition. This message exchange is traditionally specified, modelled and executed by means of some flow language like BPEL or WSCL , or can be represented as a workflow using YAWL , or can be modelled by means of a FSM [26]. Based on the pragmatic ideas described in [19] a the set of measures reported in [20], in this section a set of similarity measures are designed and adapted for comparing Web service composition models. Formally a Web service composition C is represented as follows.

$$C = (S, s_0, A, \delta, F),$$

Where

S , is the set of states
s_0, is the initial state
A is the set of service operations to be processed by C
δ, is the transition relation, $\delta : S \times A \rightarrow S$
F, is the set of final states

States in a Web service composition model represent the possible situations that can occur before and after the invocation of an operation, and are defined by the set of input and output parameters. ***Transitions*** in Web service compositions work over the cartesian product of the sets of states and operations. In particular, the Web service compositions modelled in this work are not deterministic. That is, given a pair of state and message (s, a) the transition $\delta (s, a)$ leads to different states, therefore transitions are modelled as a mathematical relation. By representing a Web service composition as a FSM, the states represent the possible situations of the execution. The initial state s_0, represents the beginning of the invocation sequence of operations of a Web service composition; a intermediate state s_i is designed to be achieved after the execution of the service operation a_j, and $s_f \in F$ is a final state achieved after the invocation the last service operation.

3.1. State similarity

In this paper, the state similarity incorporates more useful information. In particular, in Web service compositions the situation of a given Web service invocation is defined by the set of input parameters and the returned output parameters. A transition between states in a FSM can be seen as a situation *s* that change into *s'* by processing the input *a*. In a given Web service composition, *s'* represents the resulting situation after the execution of the service operation *a*. In this work, a given Web service state is measured using the state name, state type, output parameters and service domain type.

State Name . Let s_1, s_2 be two states from different service compositions, $STokens_1$ and $STokens_2$ are the set of lexical tokens extracted from each of the names s_1, s_2, respectively. The lexical similarity between s_1, s_2 is calculated by:

$$StateNameSim(s_1, s_2) = \left| STokens_1 \cap STokens_2 \right| \; / \; \left| STokens_1 \cup STokens_2 \right| \tag{1}$$

The state name similarity measure is a value in [0, 1], where a 0 value represents a total lexical difference, and 1 represents full similarity between the names of states.

State Type. The type of the state identifies its execution position throughout the service composition. Accordingly, there are three types of states: *starting*, *intermediate* and *final*. In order to obtain this information, let $type(s_i)$ be the function that reads and returns the state type of a particular state. Let $type(s_1)$, and $type(s_2)$ be two state types from different service compositions. The state type difference is defined by:

$$TypeDif((type(s_1), type(s_2))) = \begin{cases} < 0 & \textit{if } type(s_1) < type(s_2) \\ > 0 & \textit{if } type(s_1) > type(s_2) \\ 0 & \textit{if } type(s_1) = type(s_2) \end{cases} \tag{2}$$

Let s_1, s_2, be two states from different service compositions. The state type similarity between s_1, s_2, $StateTypeSim(s_1, s_2)$ is calculated by:

$$StateTypeSim(s_1, s_2) = \begin{cases} 1, & \textit{if } type(s_1) = type(s_2) \\ 1/\left| TypeDif(s_1, s_2) \right| & \text{otherwise} \end{cases} \tag{3}$$

The state type similarity is a value in [0, 1], where 1 sets a total type similarity between the states.

Output Parameters . The output of the Web service operation invocation is used to measure its structural information, using parameter names and parameter data types. Let $s_1 = (sname_1, Cp_1)$, and $s_2, = (sname_2, Cp_2)$ be two states from different service compositions, where $Sname_m$ is the state name and Cp_m is the set of output parameters of state *m*. Each parameter *i* is defined by a pair of name $nameP_i$ and data type $typeP_i$. Their respective sets of output parameters are defined as follows:

$$Cp_1 = \{ (nameP_1, typeP_1), (nameP_2, typeP_2), \ldots , (nameP_i, typeP_i)\},$$
$$Cp_2 = \{ (nameP_1, typeP_1), (nameP_2, typeP_2), \ldots , (nameP_i, typeP_i)\}.$$

Parameter similarity between output parameters is defined as the ratio of the intersection divided by the union of both sets of output parameters.

$$ParamsSim(s_1 , s_2) = |Cp_1 \cap Cp_2| \;/\; |Cp_1 \cup Cp_2| \qquad (4)$$

The output parameter similarity measure is a value in [0, 1], where a 0/1 represents a total difference/similarity between parameters.

Service Type . This data relates the service operation with its application domain. This information is relevant, because it allows using the semantic information of the service and associating the domain of the service with its result. To obtain this information, let $WStype(s_i)$ be a program function that reads and returns the state service type. Let $WStype(s_1)$, $WStype(s_2)$, be two service state types from different service compositions. The service domain type similarity is calculated by:

$$WSTypeDif(s_1 , s_2) = \begin{cases} <0, & if \;\; WStype(s_1) < WStype(s_2) \\ >0 & if \;\; WStype(s_1) < WStype(s_2) \\ 0, & if \;\; WStype(s_1) = WStype(s_2) \end{cases} \qquad (5)$$

Let s_1, s_2, be two states from different service compositions. The state service type similarity is calculated as follows:

$$StateWSTypeSim(s_1, s_2) = \begin{cases} 1, & if\ WStype(s_1) = WStype(s_2) \\ 1/|WSTypeDif\,(s_1, s_2)| & \text{otherwise} \end{cases} \qquad (6)$$

The service type similarity measure is a value in [0, 1], where 1 sets a total service type similarity between the states.

State Similarity . The general state similarity is calculated by the mean of state name lexical similarity, state type similarity, output parameter similarity and service type similarity.

$$\begin{aligned} StateSim(s_1 , s_2) = \\ StateNameSim(s_1 , s_2) + \\ StateTypeSim(s_1 , s_2) + \\ ParamsSim(s_1 , s_2) + \\ StateServiceTypeSim(s_1 , s_2) \;\; / \; 4 \end{aligned} \qquad (7)$$

The state similarity measure is a value in [0, 1], where a 0/1 represents a total difference/similarity between the states.

3.2. Message Similarity

Messages in a service composition are invocations of service operations. Let A_1, A_2 be the set of operations of a service composition C_1 and C_2 respectively. A measure of the degree of message similarity between two compositions is given.

Let $\delta_1(preS_1, a_1) \rightarrow posS_1$, and $\delta_2(preS_2, a_2) \rightarrow posS_2$ be transition functions from Web service compositions C_1 and C_2 respectively. Notice that a_1 and a_2 denote respective message at $preS_1$,

$preS_2$ of transitions, and $posS_1$, $posS_2$ the resulting states. Let p be a threshold of acceptance. A relation of similarity between messages a_1 and a_2 is established if and only if:

1) $StateSim(preS_1, preS_2) > p$,
2) $StateSim(posS_1, posS_2) > p$,

The message similarity is calculated as the mean of the state similarity between current states and the state similarity between posterior states.

$$MessageSim(a_1, a_2) = (\ StateSim(preS_1, preS_2) + StateSim(posS_1, posS_2)\ )\ /\ 2 \qquad (8)$$

3.3. Trace Similarity

The concept of *trace* was defined by Rabinovich [15] as follows: let C be a chart and s_0 be its initial state. A *trace* of C is a sequence $a_1...a_n$ of messages such that there exist nodes $s_1...s_n$ in C and $s_{i-1} \rightarrow s_i$ for $i = 1...n$.

Considering this definition in this paper a *trace* is defined as the sequence of *state – transition – state*, starting from the initial state and ending in any final state. The formal notation of a FSM trace is represented by $s_0, a_1, s_1, a_2, \ldots, s_f$.

The *trace equivalence* definition defined by Tan et al. [16] states that a trace equivalence relation between two states p and q, written $p \approx_{\mathrm{tr}} q$, holds if and only if $Tr(p) = Tr(q)$. Given two Labelled Transition Systems S and M with initial states s_0 and m_0 respectively, they say that M is trace-equivalent to S, written $M \approx_{\mathrm{tr}} S$, of only if $m_0 \approx_{\mathrm{tr}} s_0$.

Based on the concepts of *trace* and *trace equivalence*, in this paper *trace similarity* is defined as follows: Let T_1 be the set of all possible traces extracted from a given composition C_1, and T_2 the set of all possible traces from a composition C_2, C_1 and C_2 are said to be equivalent if they can execute exactly the same traces $T_1 = T_2$. Based on this trace equivalence definition, it is possible to define a measure of the degree of trace similarity between two compositions.

Let $C_1 = (S_1, s_0, A_1, \delta_1, F_1)$ and $C_2 = (S_2, t_0, A_2, \delta_2, F_2)$ be two Web services compositions, and $f_1 \in F_1$ and $f_2 \in F_2$.

Let T_1 and T_2 the set of traces extracted from C_1 and C_2 respectively, and $v \in T_1$, $w \in T_2$. Let p be a threshold of acceptance.

A relation of trace similarity between v and w is established if and only if the following conditions hold:

$$StateSim(s_0, t_0) > p,$$
$$StateSim(f_1, f_2) > p, \text{ and}$$
$$\delta_1(s_0, v) \rightarrow f_1, \text{ and } \delta_2(t_0, w) \rightarrow f_2.$$

Trace similarity is calculated as the mean of the state similarity between the initial states and the state similarity between final states.

$$TraceSim(v, w) = (\ StateSim(s_0, t_0) + StateSim(f_1, f_2)\ )\ /\ 2 \qquad (9)$$

4. EXPERIMENTATION

To test and evaluate the set of measures defined, a software tool for experimentation and evaluation of results was developed. Figure 1 shows the class diagram of the similarity measures tool implemented with Java classes to support the representation and measurement of Web service compositions.

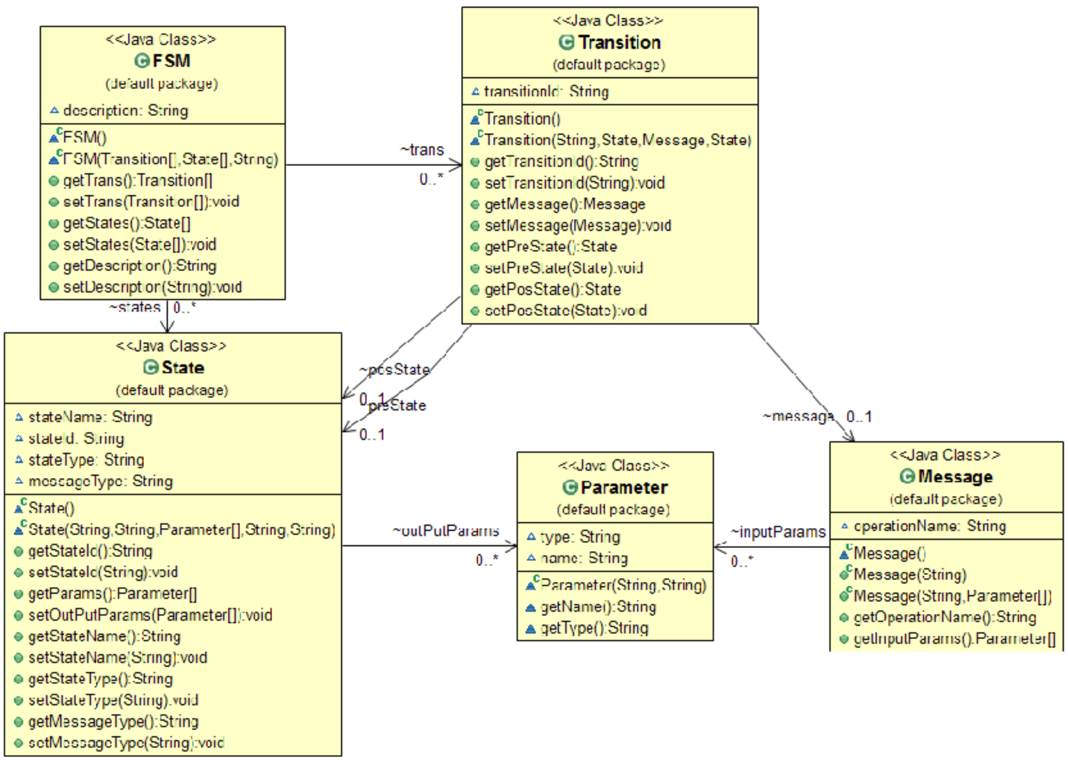

Figure 1. Class diagram of the similarity measurement tool

A service composition is a FSM that integrates states, transitions and operations. State class represents state objects of any Web service composition implemented by a set of attributes: state name, state type, message type, state id, and a set of output parameters that result by a service execution. Transition class is helpful for the representation of states connections by instantiating a current state, an operation and the posterior state. Action class is used to instantiate service operations and their associated set of contextual parameters, which are the set of input parameters required for the service execution. The Parameter class represents instances of input and output parameters, each defined by its name and data type.

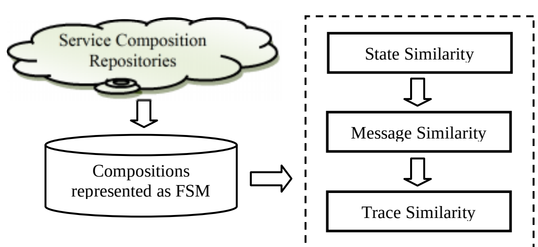

Figure 2. Experimentation methodology

The procedure for experimentation is depicted in Figure 2. First, service compositions are represented as FSM in the software model implemented. Once the service compositions are loaded in memory, states similarity measures are executed to discover similar states over a threshold of $p = 0.6$ in this case. Then message similarities are calculated for each pair of service operations defined in all transitions. Then a recursive trace tracking algorithm is executed, which identifies and extracts all the possible traces from each service composition. Finally, using a map of similar states and traces, trace similarity is calculated to discover similar traces under a certain threshold.

4.1. Representation of Web Service Compositions as FSM

Web services compositions identified by C_1, C_2 and C_3 depicted in Figures 3, 4 and 5 were represented as FSM.

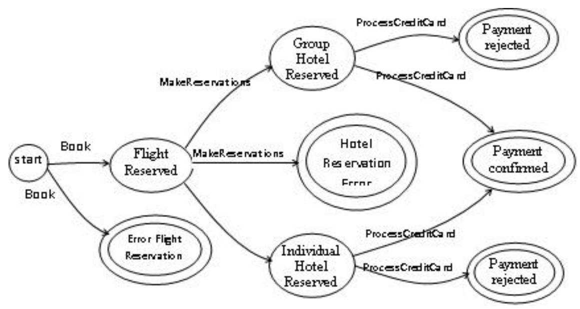

Figure 3. FSM of composition C_1

$C_1 = (S_1, s_0, A_1, \delta, F)$,

Where

S_1 = {"Flight Reserved", "Group Hotel Reserved", "Individual Hotel Reserved", "Error Flight Reservation", "Hotel reservation Error", "Payment Rejected", "Payment Confirmed"}

A_1 = {"Book", "MakeReservations", "ProcessCreditCard"}

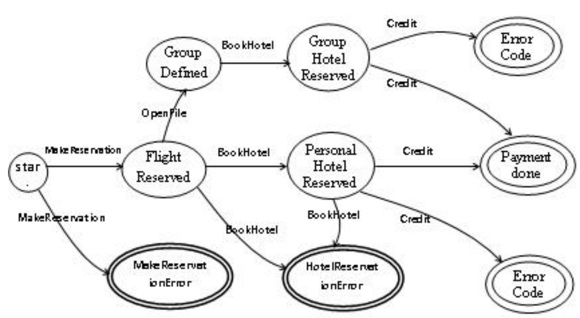

Figure 4. FSM of composition C_2

$C_2 = (S_2, s_0, A_2, \delta, F)$,

Where

S_2 = {"Flight Reserved", "Group Defined", "Group Hotel Reserved", "Personal Hotel Reserved", "Make Reservation Error", "Hotel Reservation Error", "Payment Done", "Error Code"}
A_2 = {"MakeReservations", "OpenFile", "BookHotel", "Credit"}

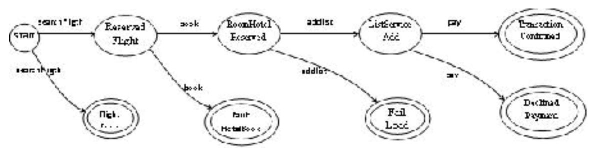

Figure 5. FSM of composition C_3

$C_3 = (S_3, s_0, A_3, \delta, F)$,

Where

S_3 = {"Reserved Flight ", "Room Hotel Reserved", "ListService Add", "Transaction Confirmed", "Flight Error", "Fault Hotel Book", "Fail Load", "Declined Payment"}
A_3 = {"searchFlight", "book", "addList", "pay"}

State-based Similarity Measures

In order to obtain state similarity between compositions, each composition must be compared with every different composition. Therefore, the number of different pairs to be compared is 3. For each pair of different compositions their sets of states are compared each other using Formula (7) to find state similarities above a threshold of 0.6 (see Figure 6).

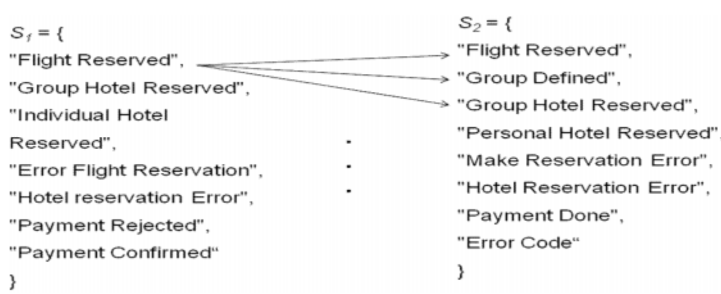

Figure 6. Process for comparing sets of states from two compositions.

For each pair of states their *StateName*, *StateType*, *ParamSim* and *ServiceType* measures are calculated and the final state similarty is obtained. Those states that resulted in a similarity with a threshold > 0.6 are shown in Table 1, Table 2 and Table 3, respectively. Table 1 shows the results after comparing the sets of states from compositions (C_1, C_2), which are .

Table 1. State similarity between compositions C_1 and C_2

States from composition C_1	States from composition C_2	Similarity
start	*start*	0.8125
FlightReserved	*FlightReserved*	0.7955
ErrorFlightReservation	*MakeReservationError*	0.6750
GroupHotelReserved	*GroupHotelReserved*	0.8250
GroupHotelReserved	*PersonalHotelReserved*	0.7083
IndividualHotelReserved	*GroupHotelReserved*	0.7000
IndividualHotelReserved	*PersonalHotelReserved*	0.7500
HotelReservationError	*HotelReservationError*	0.8750
PaymentConfirmed	*TransactionConfirmed*	0.6833
PaymentRejected	*PaymentErrorCode*	0.6875

Table 2. State similarity between compositions C_1 and C_3

States from composition C_1	States from composition C_3	Similarity
start	*start*	0.8214
FlightReserved	*reservedFligth*	0.6333
ErrorFlightReservation	*fligthError*	0.6125
GroupHotelReserved	*roomHotelReserved*	0.6250
IndividualHotelReserved	*roomHotelReserved*	0.6250
HotelReservationError	*faultHotelBook*	0.6750
PaymentConfirmed	*transactionConfirmed*	0.6458
PaymentRejected	*declinedPayment*	0.7083

Table 3. State similarity between compositions C_2 and C_3

States from composition C_2	States from composition C_3	Similarity
start	*start*	0.8056
FlightReserved	*reservedFligth*	0.6083
MakeReservationError	*fligthError*	0.6875
GroupHotelReserved	*roomHotelReserved*	0.6477
PersonalHotelReserved	*roomHotelReserved*	0.6250
HotelReservationError	*faultHotelBook*	0.6750
TransactionConfirmed	*transactionConfirmed*	0.7778
PaymentErrorCode	*declinedPayment*	0.6125

Message similarity is calculated for each different pair of service compositions. To calculate Message Similarity first the sets of transitios of each service composition is obtained, then for each transition its message is used to calculate message similarity is calculated. Figure 7 ilustatrates an example of comparing the message "*Book*" from composition C_1 with the message "*MakeReservations*" from composition C_2.

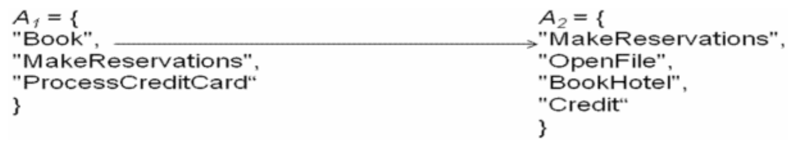

Figure 7. Process for comparing sets of transitions from different compositions.

Using Formula 8 the calculation of message similarity between "*Book*" and "*MakeReservation*" is obtained as follows:

1. *MessageSim*(a_1, a_2) = (*StateSim*(*pre*S_1, *pre*S_2) + *StateSim*(*pos*S_1, *pos*S_2)) / 2
2. *MessageSim("Book", "MakeReservations") = (StateSim("start", "sart") + StateSim("Flight Reserved", "Flight Reserved")) / 2*
3. *MessageSim("Book", "MakeReservations") = (0.8125 + 0.7955)) / 2*
4. *MessageSim("Book", "MakeReservations") = 0.804*

This calculation is executed for all messages between composition pairs. If message similarity results > 0.6 then both messages are defined to be similar. Table 4 shows the resulting pairs of similar messages.

Table 4 shows the results after comparing transitions from compositions C_1, C_2 and C_3.
Table 4. Message similarity between compositions

Similar messages from compositions C_1 and C_2	
Book	*MakeReservations*
MakeReservations	*OpenFile*
MakeResrvations	*BookHotel*
ProcessCreditCard	*Credit*
Compositions C_1 and C_3	
Book	*searchFlight*
MakeReservations	*book*
Compositions C_2 and C_3	
MakeReservations	*searchFlight*
BookHotel	*book*

In order to discover pairs of similar traces between compositions (C_1, C_2), (C_1, C_3) and (C_2, C_3), their respective traces are generated, calculate state similarities between all initial states for each pair of compositions over a threshold 0.6, calculate state similarities between final states from the different compositions pairs; and finally, confirm if the sequences started in similar initial states end up in similar final states.

Considering the following traces from compositions C_1 and C_2 respectively modelled in the form of *state – message – state*

v = start - *Book FlightReserved* - *MakeReservations* - *HotelReservationError*
w = start - *MakeReservations* - *FlightReserved* - *OpenFile* - *GroupDefined* - *BookHotel* - *GroupHotelReserved* - *BookHotel* - *HotelReservationError*

Trace similarity between both traces is calculated as follows:

1. $TraceSim(v, w) = (StateSim(s_0, t_0) + StateSim(f_1, f_2))\ / 2$
2. $TraceSim(v, w) = (StateSim("start", "start") + StateSim("HotelReservationError", "HotelReservationError"))\ / 2$
3. $TraceSim(v, w) = (0.8125 + 0.8750)\ / 2$
4. $TraceSim(v, w) = 0.8437$

This calculation is executed for all traces between composition pairs. If trace similarity results > 0.6 then both traces are defined to be similar. Table 5 shows the resulting pairs of similar traces.

Table 5. Some of resulting similar traces between compositions

Traces	**Sequence from composition C_1**	**Sequence from composition C_2**
v_2, w_2	*start Book FlightReserved MakeReservations HotelReservationError*	*start Make Reservations Flight Reserved Open File Group Defined Book Hotel Group Hotel Reserved Book Hotel Hotel Reservation Error*
v_3, w_2	*start Book FlightReserved MakeReservations GroupHotelReserved MakeReservations HotelReservationError*	*start Make Reservations Flight Reserved Open File Group Defined Book Hotel Group Hotel Reserved Book Hotel Hotel Reservation Error*
v_4, w_3	*start Book FlightReserved MakeReservations IndividualHotelReserved MakeReservations HotelReservationError*	*start MakeReservations FlightReserved BookHotel PersonalHotelReserved BookHotel HotelReservationError*
v_5, w_5	*start Book FlightReserved MakeReservations GroupHotelReserved ProcessCreditCard PaymentConfirmed*	*start MakeReservations FlightReserved OpenFile GroupDefined BookHotel GroupHotelReserved Credit TransactionConfirmed*
v_8, w_7	*start Book FlightReserved MakeReservations IndividualHotelReserved ProcessCreditCard PaymentRejected*	*start MakeReservations FlightReserved OpenFile GroupDefined BookHotel GroupHotelReserved Credit PaymentRejected*
Traces	**Sequence from composition C_1**	**Sequence from composition C_3**
v_4, x_2	*start Book FlightReserved MakeReservations IndividualHotelReserved MakeReservations HotelReservationError*	*start searchFligth reservedFligth book faultHotelBook*
v_6, x_4	*start Book FlightReserved MakeReservations IndividualHotelReserved ProcessCreditCard PaymentConfirmed*	*start searchFligth reservedFligth book roomHotelReserved addlist listServiceAdd pay transactionConfirmed*
v_8, x_5	*start Book FlightReserved MakeReservations IndividualHotelReserved ProcessCreditCard PaymentRejected*	*start searchFligth reservedFligth book roomHotelReserved addlist listServiceAdd pay declinedPayment*
Traces	**Sequence from composition C_2**	**Sequence from composition C_3**
w_2, x_2	*start MakeReservations FlightReserved OpenFile GroupDefined BookHotel GroupHotelReserved BookHotel HotelReservationError*	*start searchFligth reservedFligth book faultHotelBook*
w_5, x_4	*start MakeReservations FlightReserved OpenFile GroupDefined BookHotel GroupHotelReserved Credit TransactionConfirmed*	*start searchFligth reservedFligth book roomHotelReserved addlist listServiceAdd pay transactionConfirmed*
w_7, x_5	*start MakeReservations FlightReserved OpenFile GroupDefined BookHotel GroupHotelReserved Credit PaymentRejected*	*start searchFligth reservedFligth book roomHotelReserved addlist listServiceAdd pay declinedPayment*

5. EVALUATION

To evaluate experimental results, *Precision* and *Recall* measures are used [12]. The *Precision* of *state similarity* is defined as the number of relevant (correct) similar states retrieved with a measure result >= 0.6, divided by the total number of states pairs retrieved. And *Recall* is defined as the number of relevant similar states with a measure result >= 0.6, divided by the total number of correct relevant similar states pairs (which should have been selected). To evaluate message similarity results, *Precision* is defined as the number of relevant (correct) similar message pairs retrieved with a measure result >= 0.6, divided by the total number of message pairs retrieved. Results of *Precision* and *Recall* are shown in Table 6. To evaluate trace similarity results, *Precision* is defined as the number of relevant (correct) similar trace pairs retrieved, divided by the total number of trace pairs retrieved.

Table 6. Precision, recall and F-measures results

	State similarity			**Message similarity**			**Trace similarity**		
Pair	Precision	Recall	F measure	Precision	Recall	F measure	Precision	Recall	F measure
c1, c2	0.80	1.00	0.90	0.75	1.00	0.88	1.00	1.00	1.00
c1, c3	1.00	1.00	1.00	1.00	0.67	0.83	1.00	0.89	0.94
c2, c3	1.00	1.00	1.00	1.00	0.67	0.83	1.00	0.86	0.93

Table 6 shows *Precision* and *Recall* results of measures used for discovering similarities between service compositions. Notice that these results are highly precise because the similarity threshold is tuned up to achieve the best balance. Actually, a relevant issue is the possibility to adapt the similarity measures to any set of Web service compositions, that is, by means of a threshold tuning, regarding the particular characteristics of Web services domain. Therefore, a general method for similarity measuring is enabled.

CONCLUSIONS

This paper reports a set of similarity measures aiming at addressing the problem of finding and selecting Web service compositions that are similar with an initial composed service specification. The set of similarity measures are defined for the comparison of compositions modeled as FSM, where states represent the possible situations that can occur before and after the invocation of a service operation, and situations are defined by the set of input and output parameters. Transitions in Web service compositions work over the cartesian product of the sets of states and operations. In particular, the Web service compositions modeled in this work are not deterministic. That is, given a pair of state and message (s, a) the transition $\delta\,(s, a)$ leads to different states, therefore transitions are modeled as a mathematical relations.

A tool for experimentation was implemented which starts calculating state similarities, which means identifying similar situations between all compositions. Then calculates message similarities between all compositions, this step allows identification of similar service operations considering their input and output parameter sets. Finally, using state similarities and message similarities trace similarity is calculated leading to the recognition of similar service execution paths, a necessary and preliminary calculation towards the selection of a similar behaving composition. Results of these calculations are promising according with the precision and recall evaluation measures.

The set of proposed similarity measures that have been presented in this work allow the progressive comparison of service compositions. These comparisons are made in granular form and evaluating different elements of the compositions, so that it is possible to know numerically the syntactic differences they have and at the time of selecting a similar composition to know in advance the required changes.

Next step of this work is to develop an experimentation tool to allow remote users to publish Web service composition models, to compare service models using similarity measures and to select the compositions that best suit their needs. Another important research topic derived from the set of similarity measures is their application to classification and clustering algorithms.

REFERENCES

[1] Jeong, B., Cho, H., Lee, C. On the functional quality of service (FQoS) to discover and compose interoperable Web services. International Journal of Expert Systems with Applications, 2008, pp. 5411-5418.
[2] Hamming, R. Error detecting and error correcting codes. Bell System Technical Journal, Vol. 29 No. 2, 1950, pp. 147-160.
[3] Levenshtein, V. I. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady Vol. 10, 1966, pp. 707-10.
[4] Plebani, P., & Pernici, B. URBE: Web Service Retrieval Based on Similarity Evaluation, IEEE Transactions on Knowledge and Data Engineering, 2009, Vol. 21 No. 11.
[5] Dong, X., Halevy, A., Madhavan, J., Nemes, E., & Zhang, J. Similarity Search for Web Services, Proceedings of the 30th VLDB Conference, 2004, pp. 372-383.
[6] Bruno, M., Canfora, G., Di Penta, M., & Scognamiglio, R. An Approach to support Web Service Classification and Annotation. IEEE International Conference on e-Technology, e-Commerce and e-Service, 2005.
[7] Stroulia, E., Wang, Y. Structural and Semantic Matching for Assesing Web Service Similarity, International Journal of Cooperative Information Systems, 2005, Vol. 14 No. 4.
[8] McIlarith, S., Cao Son, T., & Zeng, H. Semantic Web Services. IEEE Intelligent Systems. 2001.
[9] Grigori, D., Corrales, J., Bouzeghoub, M. Behavioral matchmaking for service retrieval, IEEE International Conference on Web Services. 2006.
[10] Dumas, M., García-Bañuelos, L., Dijkman, R. Similarity Search of Business Process Models, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, Vol. 32 No. 3, 2009, pp. 23-28.
[11] Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S., & Zave, P. Matching and Merging of Statecharts Specifictaions, Proceedings of the 29th International Conference on Software Engineering, 2007.
[12] Baeza-Yates, Ribeiro-Neto. Modern Information Retrieval, New York ACM Press, Addison-Wesley. 1999.
[13] Munindar, P. Singh. The Pragmatic Web: Preliminary thoughts. NSF-OntoWeb Workshop on Database and Information Systems Research for Semantic Web and Enterprises, 2002, pp. 82-90.
[14] Sharma, V., Kumar, M. Comparative Analysis of IR Based Web Service Similarity Measures Using Vector Space Model, Global Trends in Information Systems and Software Applications, Communications in Computer and Information Science, 270, 2012, pp. 752-760.
[15] Rabinovich, A. A complete axiomatisation for trace congruence of finite state behaviours. Mathematical Foundations of Programming Semantics, Lecture Notes in Computer Science, Vol. 802, 1994, pp. 530-543.
[16] Tan, Q., Petrenko, A., Luo, G. & Bochmann, G. Testing Trace Equivalence for Labeled Transition Ssystems, Technical Report 976, Dept. of I.R.O., University of Montreal, 1995.
[17] Fethallah, H., Mine, M. Automated Retrieval of Semantic Web Services: A Matching Based on Conceptual Indexation, The International Arab Journal of Information Technology, Vol. 10 No. 1, 2013.
[18] Rinderle-Ma, S., Reichert, M., Jurisch, M. Equivalence of Web Services in Process-Aware Service Compositions, IEEE International Conference on Web Services, 2009, pp. 501-508.

[19] Alvarado, M., & Bravo, M. Pragmatic Similarity for Web Services Composition, Unpublished Manuscript, 2011.

[20] Bravo, M., & Alvarado, M. Similarity Measures for Substituting Web Services. In P. Hung (Ed.), Web Service Composition and New Frameworks in Designing Semantics: Innovations, 2012, (pp. 143-170). Hershey, PA: Information Science Reference.

Authors

Maricela Bravo is a researcher at the Systems Department of the Autonomous Metropolitan University UAM (Azcapotzalco, Mexico) since 2011. She holds a BS in Information Systems, a MSc in Computer Science and a PhD in Computer Science from CENIDET (2003 and 2006 respectively). Her research interests include Semantic Web services, Semantic Web services complex tasks support (search, discovery, match, substitute, and compose); and Ontology design.