Int J Software Informatics, Volume 7, Issue 2 (2013), pp. 309–330 Tel: +86-10-62661040
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org

Email: ijsi@iscas.ac.cn, ijsi2007@gmail.com

Web Service Choreography: Unanimous Handling of Control and Data

Syed Adeel Ali[1], Partha Roop[1] and Ian Warren[2]

[1](Department of Electrical and Computer Engineering, The University of Auckland, New Zealand)
[2](Department of Computer Science, The University of Auckland, New Zealand)

Abstract To provide an effective service-oriented solution for a given business problem, it is necessary to explore all available options for providing the required functionality while ensuring a flawless data transfer within the composed services. Existing service composition approaches fall short of this ideal, as functional requirements and data mediation are not considered in a unified framework. We propose a service composition framework that addresses both of these aspects by integrating existing techniques in formal methods, service oriented computing and data mediation. Our framework guarantees the correct interaction of services in a composition by verifying certain behavioral constraints, and resolving data mismatches at semantic, syntactic and structural levels, in a unified manner. A tableau based algorithm is used to generate and explore compositions in a goal-directed fashion that proves or disproves the existence of a service choreographer. Data models, to detect and resolve data mismatches, are generated using WSDL documents and regular expressions. We also apply our framework to examples adapted from the existing service composition literature that provide strong testimony that the approach can be effectively applied in practical settings.
Key words: web service composition; choreography; web service control flow; data mismatches in web services

Ali SA, Roop P, Warren I. Web service choreography: Unanimous handling of control and data. *Int J Software Informatics*, **Vol.7, No.2 (2013): 309–330. http://www.ijsi.org/1673-7288/7/i159.htm**

1 Introduction

Services are the fundamental units of service oriented computing (SOC), and exist as software programs with design attributes specific to the requirements of a service-oriented architecture. A service is a network and platform independent operation to be invoked by other services or clients, and needs to overtly define its properties in a standard format to operate in a service oriented environment. For this purpose, SOC provides three native facilities: description, discovery, and communication[22].

The areas of active research in SOC include service publishing, service discovery, service selection, service composition, service execution and service monitoring. Out of these, this research work is focused on service composition which addresses the problem of efficient and effective integration of heterogeneously developed services to

Corresponding author: Syed Adeel Ali, Email: sali080@aucklanduni.ac.nz
Received 2012-10-12; Revised 2013-02-09; Accepted 2013-04-19.

produce new desired services. Service composition offers application development on top of SOC's native facilities[22]. Salient features of such applications include rapid deployment, reuse possibilities and seamless access for users to a variety of complex services. Service composition is desired in the scenarios when there is no capable service available to fulfill the given requirement, but the requirement can be met by selecting and integrating (*composing*) multiple services.

With the growth in SOC popularity, the number of available services have been increased rapidly, and it has become a necessity to automate (or semi-automate) their composition. Since the automated composition process involves several independent services talking to each other, the correct interaction among the participating services becomes vital. This correct interaction can be guaranteed by controlling the message sequence and data flow while composing the services together. Other important characteristics that a composition approach should aim to support are connectivity, quality of service and scalability[22]. Industrial standards like BPEL and WS-CDL offer support to connectivity, QoS and scalability, but do not provide any direct solution for the correctness i.e., design-time verification of web service compositions. This is the reason behind the introduction of formal methods in the field of web service composition[37].

Traditional web services are XML-based, and are termed as syntactic web services while another class of web services that are ontology-based are called semantic web services. Consequently, the composition approaches can be classified as syntactic and semantic web service compositions. Orchestration and Choreography are the two main approaches in the area of syntactic web service composition[37]. The control flow among services in a service oriented architecture can be seen from a *global perspective* via choreography, or from the view of a *single participant* (or a central coordinator) using orchestration. This research work deals with the composition of syntactic web services, where the desired functionality is realized by synthesizing a choreographer.

1.1 The driving problem

The calling of web services by web services - instead of humans - in an automated web service composition raises exciting prospects and various challenges. One of the prime challenges is to guarantee the correct interaction of independent services since this interaction, due to its message-passing nature, may lead to many subtle errors like deadlocks and incompatible behaviors. According to a survey[37], the lack of capability of the industrial approaches, for the design time verification of service compositions, welcomes formal methods in this domain.

The correctness constraint of a composition depends upon the control flow and data mismatches among independent, communicating software pieces. Control flow is very much related to the functionality i.e., the sequencing of service calls in a specified manner to obtain a new service with desired behavior. This can be done by synthesizing an orchestrator or a choreographer that could guide the services to invoke each other in a manner that ensures the desired behavior.

Data mismatches occur when the input-output entities of interacting services do not match. For example, one service may require the *surname* of a person while the record keeping service returns the *family-name* upon request. Data mismatches can be classified as systematic, syntactic, structural, and semantic[31]. Along with

the control flow, the data transferred among the participating web services for a composition are clearly very important for correct composition. Nevertheless, most of the existing approaches either neglect data, or need very small ranges to be related to the data types.

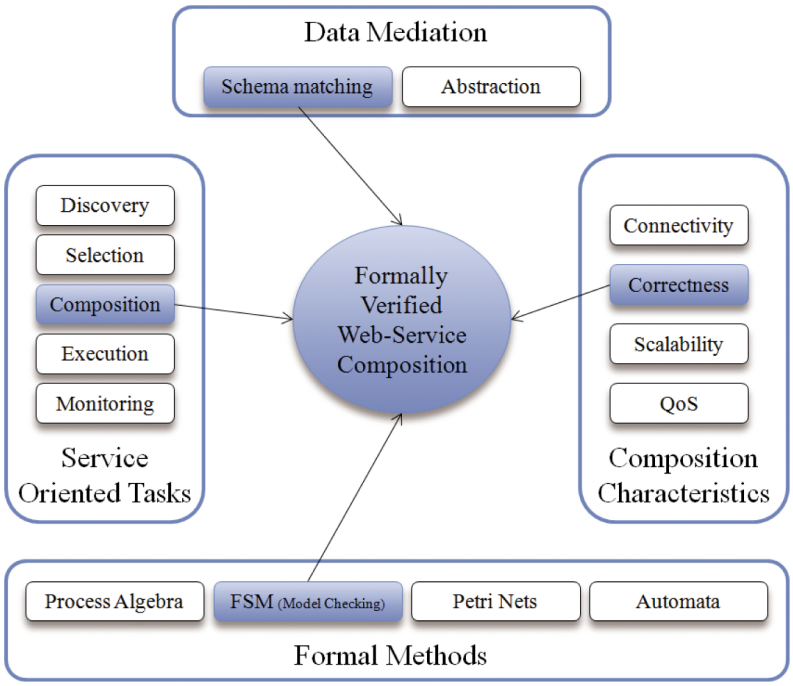

Figure 1. Shaded boxes present the adopted techniques among the existing ones

1.2 Our solution and contributions

We develop a web service composition solution that guarantees a correct-by-construction composition by managing the control and data flow among web-services. Our formal approach is based on model checking, where web services are represented as Synchronous Kripke Structures[35], while the desired functional behavior or control flow is specified with the help of CTL[12] properties. A model verification algorithm[36] has been extended to generate and explore compositions in a goal-directed fashion that proves or disproves the existence of a choreographer. We use WSDL documents[29] of web services along with regular expression to generate data models to detect and resolve data mismatches during choreographer synthesis.

In a previous attempt[1], we adopted a two-phased composition methodology. The first phase, named as *Control Phase*, was actually responsible for the composition synthesis via CTL model checking, while in the second step, termed as *Data Phase*, the composition path was traversed to check for any data mismatches. The major drawback was to check for the data mismatches after obtaining a composition path, as encountering an unresolvable mismatch in the obtained path would terminate the process, nullifying the outcome of the model checking step. This drawback is resolved in the proposed approach by performing the data matching while constructing the synchronous product of the services in the first phase.

Another advancement has been made for the input of regular expressions. Previously, the required regular expressions were provided by the service vendors, which imposes the limit of using only those services whose vendors are in contract to provide regular expressions, if needed. In the current framework, the input of regular expressions has been made a feature for the service users, and is described in section 4. The major contributions can be summarized as:

- Much improved (compared to Ref. [1]) formally verified service composition framework with unified management for control flow and data mismatches (at semantic, structural and syntactic levels) among web services.
- The technique is goal-directed, and only generates (making the approach on-the-fly) and investigates parts of possible composite behaviors that are needed to verify the specified properties and the existence of a corresponding choreographer.
- Preliminary experimental evaluation via variety of composition problems provides strong testimony of the effectiveness of the technique.

1.3 Paper organization

After reviewing the state-of-the-art in section 2, an illustrative example is presented in section 3 to exploit the problems addressed in this paper. Section 4 elucidates the fundamental concepts and specifications used, while the composition framework is presented in section 5. Section 6 discusses the obtained results. Finally, we conclude with some future directions of research in Section 7.

2 Related Work

The three primary domains of research related to this work are: formal methods, service oriented computing (SOC), and data mediation. For the proposed solution, we combine *model checking* from formal methods, *web service composition* from SOC and *schema matching* from the domain of data mediation. Moreover, our solution addresses the *correctness* of composition as the desired characteristic of service composition. This section presents an overview of the overlapped approaches in these domains while, Fig. 1 depicts the derivation of our solution from the existing techniques in the aforementioned domains.

Web service composition

Traditional web services are XML-based, and are termed as syntactic web services while another class of web services that are ontology-based are called semantic web services. Therefore, the composition approaches can be classified as syntactic and semantic web service composition.

Syntactic web service composition

Orchestration and Choreography are the two main approaches in the area of syntactic web services composition. The control flow (also known as work-flow) among services in a service oriented architecture can be seen from a global perspective via ***choreography*** or from the view of a single participant using ***orchestration***. Defining a centralized process for communicating and integrating

heterogeneous web services is a fundamental task in the orchestration of web services [10]. In contrast, the fundamental tasks of web service choreography is to check that messages are transferred in a specific sequence as well as according to an interface description[10].

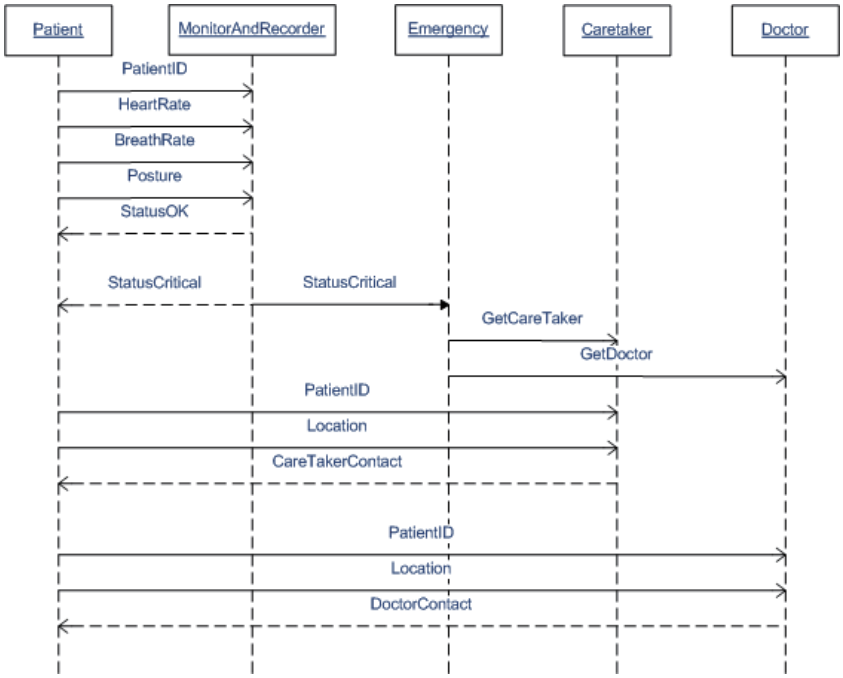

Figure 2. MyHeartCare - Sequence of operations

The difference between web service choreography and orchestration can be understood by the analogy of dance choreography and orchestration. In dance choreography, the performers produce a combined effect by synchronously acting in order to achieve the desired outcome without the involvement of a central conductor or director. The overall plan or design is given to the performers, and each performer takes responsibility for his/her own steps. On the other hand, in musical orchestra, a central coordinator is required to direct the group of performers. The performers independently play their instruments, and do not have any direct interaction among them[15].

Most of the orchestration literature focuses on the development of languages to implement service orchestrations where control as well as data flow go through a centralized server[4]. Among the existing proposals for orchestration languages, Web Services Business Process Execution Language (WS-BPEL), or in short BPEL[3], is the most widely used one. BPEL is based on XML, and uses the Web Services Description Language (WSDL)[29] for describing the web services. BPEL enables the interactions and composition of a set of web services, and describes relationships among multiple invocations via control and data flow links. Examples of the existing orchestration frameworks can be found through BPEL in the Business Process Modeling community and through Taverna[30] in life sciences community.

The second approach, web service choreography, does not rely on a central coordinator; rather it achieves the composition by peer-to-peer communication among the interacting web services. The Web Services Choreography Description Language (WS-CDL)[16] is the most popular language among the emerging choreography languages. WS-CDL is also based on XML and aims at the composition of interoperable, peer-to-peer interactions between any type of web services regardless of their programming models, supporting platforms or implementation details. Various approaches for web service choreography have been reported in literature. For example, Pathak et al.[32] addressed the issue of realizing a composite service through a parallel composition of a subset of component services. Mitra et al.[24] represented services and goals using I/O automata. In this work, all possible behaviors that could be realized from the services were identified using a simple or transducing choreographer. Synthesis of a choreographer was inferred to be possible if the goal behavior was simulated by all possible composed behavior. Later on, Mitra et al.[23] amended the work by developing a goal-directed technique that only searches and generates a subset of the possible composite behavior that is needed to prove or disprove the existence of the desired goal service and the realizability of a corresponding choreographer. Another major advancement was to have an optimum overhead for service composition, reported in Ref. [25], Ref. [26] by synthesizing a distributed choreography approach.

Semantic web service composition

Traditional web service technologies only focus on the syntactic aspects of web services, making them un-adaptable to the changing environments without human aid. On the other hand, semantic web services[21] describe the web services using explicit semantics that are machine-understandable. Process level description of web services is provided by the Semantic Web[6] that helps the evolution of the domain in a logical manner, and the domain concepts shared among web services are formalized using ontologies. OWL (Web Ontology Language)[20] and WSMO (Web Service Modeling Ontology)[34] are the two main approaches in the field of semantic web services. Both approaches aim to provide standardized semantic description of web services.

Formal methods for service composition

A major problem of the industrial standard approaches, the absence of software tools to verify the correctness of web service composition, is the main driver for using formal methods. Particularly, formal methods and tools can be helpful to decide whether web services and their composition satisfy specific desirable properties. If one should discover that a web service composition does not conform to an abstract specification, or that a main property has been violated, this can be helpful to correct a design, or to trace bugs in a service.

In the literature, several formal approaches have been reported that guarantee the correctness of web service compositions. Some approaches are already discussed in the previous section (choreography approaches). Others include model checking[13], automatic composition of finite-state machines[5], variants of automata (I/O automata[23, 24], timed automata[2] and team automata[38]) etc. The control part of our approach resembles the on-the-fly, goal-directed composition

approach[23], where web services and the goal (also a service) are represented using I/O automata. The advantage of our approach is that instead of only checking the reachabilty of the goal, it can also verify the specified behavioral constraints en-route to the goal.

Data mediation in service composition

According to a data classification, named as the S classification[8,9], data mismatches among information systems could possibly occur at four levels: systemic, syntactic, structural, and semantic. Systemic level mismatches refer to the difference in the combination of software and hardware used to implement the applications at communicating ends. Systemic mismatches have largely been solved by the advent of the standard underlying network protocols like IP, TCP and UDP, as these universally agreed-upon protocols assure successful transfer of a "sequence of bytes" regardless of the hardware or software used at both sides. This leaves us to deal with the remaining mismatch types, that occur at syntactic, structural, and semantic levels.

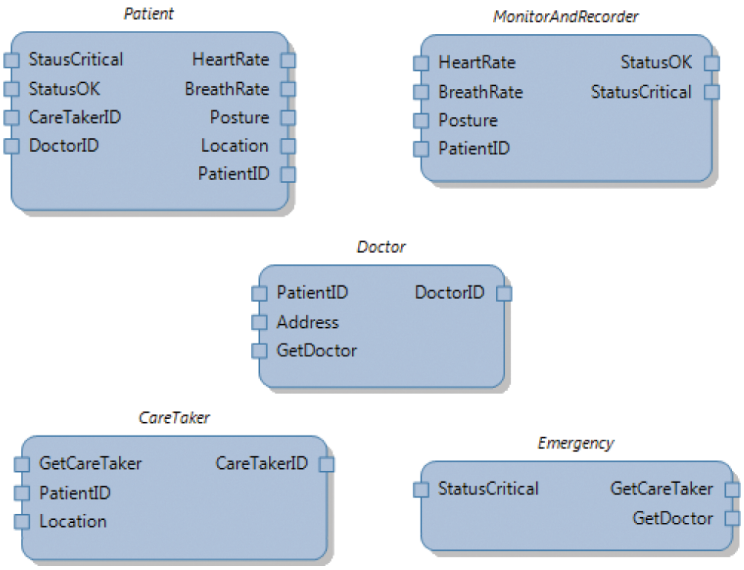

Figure 3. Web Services used in *MyHeartCare* with input and output variables

From the domain of Data Mediation, we investigate schema matching techniques to build our solution. Schema matching techniques (see Ref. [33] for a survey) can mainly be divided into two classes: (i) machine learning techniques[11], where domain experts are involved at the beginning of the process to provide information to the system, or at the end to validate the outcome; and (ii) user guided techniques[27], which proceed with the help of human intervention in an iterative manner throughout the process. Data heterogeneity solutions for web services are also found in the literature, for example the semi-automatic approach for ontology-to-ontology mediation for semantic web services[27], run-time data adapter generation for web service orchestration[28] and data aspects modeling for the correctness of the composition via abstraction techniques[17].

Another data-related operation, termed as data flow, is performed by routing the data among the ports of web services. No data transformation is performed here in case of a mismatch. An example of this is the ASTRO approach[19], where data flow requirements are collected in a hyper-graph called a *data net*, which is translated into a state transition system that becomes part of a planning domain for composition.

3 Illustrative Example

We motivate the composition approach presented in this paper using an example (taken and modified from Refs. [18, 39]), which describes *MyHeartCare* - a web service system which facilitates the post surgery monitoring of a heart patient. *MyHeartCare* relies on five independently existing services namely Patient, Doctor, CareTaker, MonitorAndRecorder, and Emergency:

- The **Patient** service transmits the vital signs of a patient at a regular interval. It also sends out other information like patient's location in case of an emergency.
- The **MonitorAndRecorder** service regularly receives and records the vital signs of the patient. It also invokes the emergency service in special circumstances.
- The **Emergency** service is responsible to call the Doctor and CareTaker services upon receiving a request from the MonitorAndRecorder service.
- The **CareTaker** service receives the patient's ID and location, and sends back the ID of the nearest available care taker. Care takers are local volunteers with some basic training to provide initial assistance to the patient, and to take the patient to the hospital if needed.
- The **Doctor** service also takes in the patient's ID and address, and sends back the ID of an available doctor at the nearest hospital. The doctor stays at the hospital, and monitors the patient online.

Service models, depicting the input and output variables are shown in Fig. 3. A scenario to elucidate the problem as well as to demonstrate our solution is described next.

Scenario:

After being discharged after a heart surgery, a desired practice is to observe the vital signs (like posture, heart rate, and breathing rate) of a patient closely and regularly. With the advent of smart phones, it is possible to equip the patient with a smart phone capable of recording those vital signs, and to regularly send it to a remote monitoring system via the **Patient** service. The **MonitorAndRecorder** service receives and records the vital signs of the patient as a normal function. In special circumstances, for example when a particular combination of the vital signs readings indicate an abnormal behavior, the remote medical staff should be notified immediately. The situation can become worse if the patient is alone, or is away from home. In those cases, the **Emergency** service is responsible to notify both the **Doctor** and **CareTaker** services. Upon receiving the emergency signal, the **CareTaker** service immediately determines the nearest available care taker using the

location information provided by the **Patient** service (via GPS of the smart phone), while the **Doctor** service determines a doctor on service at the nearest hospital. The selected care taker goes to attend the patient immediately, while the nominated doctor at hospital monitors the vital signs of the patient remotely. Upon reaching to the patient, the care taker determines if his assistance would be enough, or the patient should be taken to the hospital. The sequence diagram (Fig. 2) show the sequence of operations/service invocations in the above scenario.

Control flow:

In our scenario, there are certain conditions to be met for the desired compositional behavior. For example, the doctor service should only be sent a patient's information when there is an emergency situation i.e., the communication between the doctor and the patient services should be disabled until the emergency service sends out the emergency signal. In other words, services taking part in the composition should only call other services in a *desired sequence*. This desired sequencing of service calls is termed as the control flow, and managing the correct control flow is a major goal of the choreographer. Control flow conditions for our scenario are specified in the next section.

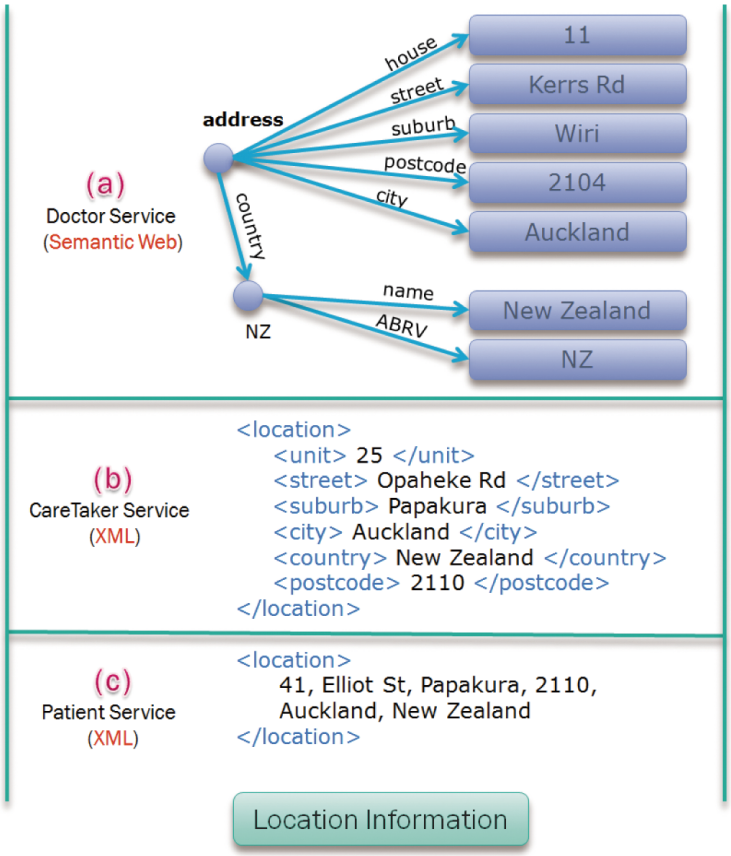

Figure 4. **Data Mismatches**: (a) and (b) or (c) shows **Semantic** Mismatch; (b) and (c) presents **Structural** Mismatch; and (a) and (b) or (c) depicts **Syntactical** Mismatch.

Data mismatches:

To appreciate the data mismatch problem, let's assume that the services taken are developed independently by different vendors. Consequently, there will be no assurance of uniformity among the semantics, structure and syntax of the inputs and outputs of the services. It can be observed that most of the operations in our scenario revolve around the current locations of the entities, such as processing the *location* of the patient to trace out the nearest available care-taker, or determining the *address* of the nearest hospital. Making use of these location entities, the following subsections present the different levels of mismatches that can occur while composing the services together.

Semantic mismatch

Semantics describe the inherent meaning of the data and its interpretation. A semantic mismatch occurs when the interacting services refer to the same piece of information with different names or synonyms. For example, the **Patient** service sends out the patient's location via variable *location* (Fig. 4(c)) to the **Doctor** and **CareTaker** services. There will not be a semantic mismatch in communicating with the **CareTaker** service, as it also uses a similarly named *location* variable. However, it can be seen in Fig. 4(a) that the **Doctor** service manipulates the location information via a variable *address* rather than *location*, which makes both the services semantically incompatible.

Syntactic mismatch

Syntax refers to the grammar or language rules of the data-type being used. A syntactical mismatch happens when the sending service outputs some data but the receiving service can not comprehend it due to the difference in the low-level representation i.e., syntax of the sent and the received data. For example, the **Patient** service uses its location data encoded in XML as depicted in Fig. 4(c), and when it wants to provide the location information to the **Doctor** service, there is a syntactic mismatch because the **Doctor** service uses the location information as notations of the Semantic Web[6] (represents data using subject-predicate-object triples, where the subject and the object act as source and destination nodes while the predicate connects both the nodes as a labeled edge) as shown in Fig. 4(a).

Structural mismatch

Structure refers to the particular constitution of the primitives within a data-type. A structural mismatch takes place when the receiving service finds the exchanged data in other-than-expected shape, style or order. For example, both the **Patient** and the **CareTaker** services use XML encoded locations for their operations. However, it can be observed from Figs. 4(b) and (c) that both the services use different sets of XML tags to represent their locations. There is only one tag in the location schema of the **Patient** service, while location field of the **CareTaker** service has been divided into several sub-tags. Another example could be having the same number of tags with different orders and semantics.

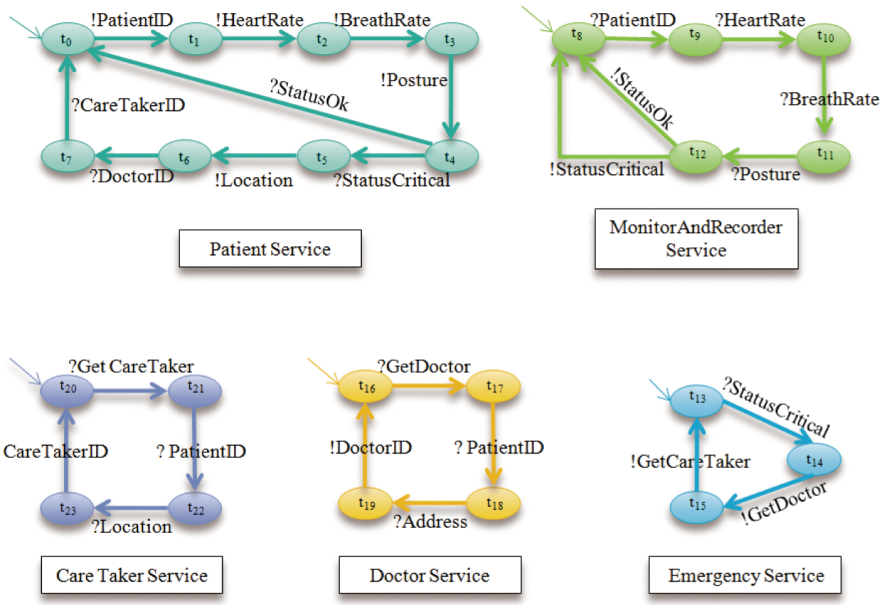

Figure 5. Formal representations of the web services as Synchronous Kripke Structures

4 Preliminaries

Carrying out web-services composition using a formal method requires (i) formal representation of the web services, and (ii) formal specifications of the goal, for controlling the message flow and to resolve data mismatches.

4.1 Services representation

4.1.1 Synchronous kripke structure

For the formal representation of web-services, the notations of Synchronous Kripke Structures[35] (SKS) has been adopted in the proposed solution.

Synchronous Kripke structure: SKS is a finite state machine represented by a tuple (AP, S, s_0, I, O, R, L) where:

- AP is a set of atomic propositions.
- S is a finite set of states.
- $s_0 \in S$ is the initial state.
- I is a finite set of inputs.
- O is a finite set of outputs.
- $R \subseteq S \times (I \cup O) \times S$ is the deterministic transition relation between the states.
- $L : S \rightarrow 2^{AP}$ is the state labeling function.

We use the notation $s \xrightarrow{?x} s'$ (resp. $s \xrightarrow{!x} s'$) to denote input (resp. output) action x when the system moves from s to s'.

Every service is manually translated from its respective service description (WSDL document), and represented as a Synchronous Kripke Structure in Fig. 5. For example, the Doctor service is represented as an SKS where:

- $AP = \{t_{16}, t_{17}, t_{18}, t_{19}\}$.
- $S = \{s_{16}, s_{17}, s_{18}, s_{19}\}$.
- $I = \{?GetDoctor, ?PatientID, ?Address\}$.
- $O = \{!DoctorID\}$.
- $R = \{(s_{16} \xrightarrow{?GetDoctor} s_{17}), (s_{17} \xrightarrow{?PatientID} s_{18}), (s_{18} \xrightarrow{?Address} s_{19}), (s_{19} \xrightarrow{!DoctorID} s_{16})\}$.
- L can be used to obtain the state-labels of any state e.g., $L(s_{16}) = t_{16}$.

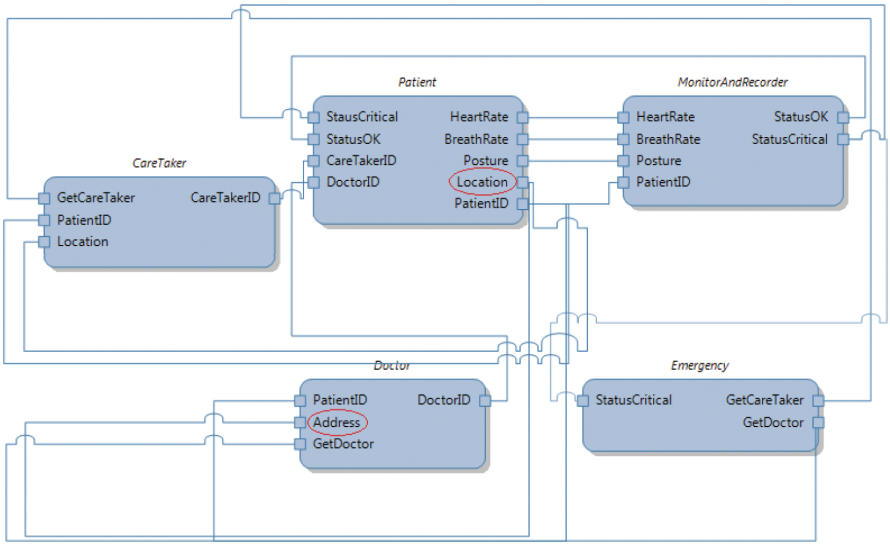

Figure 6. Connected Services. Connection between the *Location* and the *Address* fields of the Patient and the Doctor services is an example of semantic matching

4.1.2 Parallel composition of SKS

The parallel composition[36] of the services (as SKSs) describes all possible behaviors exhibited by the services via exchange of messages (output from one is consumed by input to another). A connection between two service ports in the parallel composition is made after resolving the data mismatches between the ports, if present.

Parallel Composition of SKS: A parallel composition is a synchronized product of all of the individual services. Given $SKS_1 = (AP_1, S_1, s_{0_1}, I_1, O_1, R_1, L_1)$ and $SKS_2 = (AP_2, S_2, s_{0_2}, I_2, O_2, R_2, L_2)$, their parallel composition, denoted by $SKS_{1||2}$ is $(AP_{1||2}, S_{1||2}, s_{0_{1||2}}, I_{1||2}, O_{1||2}, R_{1||2}, L_{1||2})$, where:

- $AP_{1||2} = AP_1 \cup AP_2$.
- $S_{1||2} = S_1 \times S_2$.
- $s_{0_{1||2}} = (s_{0_1}, s_{0_2})$.
- $I_{1||2} = I_1 \cup I_2$.
- $O_{1||2} = O_1 \cup O_2$.
- $R_{1||2} \subseteq S_{1||2} \times [\tau] S_{1||2}$: $(s_1 s_2) \xrightarrow{\tau} (s'_1 s'_2) \Leftarrow s_1 \xrightarrow{!x} s'_1 \wedge s_2 \xrightarrow{?x} s'_2$, where $\tau \in (I_{1||2} \cup O_{1||2})$
- $L(s_1, s_2) = L(s_1) \cup L(s_2)$,

The notation $PC_{[n]}$ is used to describe the parallel composition of n services. Each state in the parallel composition relates to one state in each web service. A transition from states in the composition corresponds to individual synchronized transitions in each service. As there are eight, five, three, four and four states in the Patient, MonitorAndRecorder, Emergency, Doctor and CareTaker web services respectively, the maximum number of states in this parallel composition for *MyHeartCare* can be: $PC_{[5]} = 8 \times 5 \times 3 \times 4 \times 4 = 1920$.

4.2 Goal specifications

The main objective of this is paper is to determine a choreographer that could guide the provided services in a *specific* manner, as described by a number of constraints. These constraints are related to (i) the control flow management for obtaining the correct composition, and with (ii) resolving any mismatch while exchanging data among the services. The following subsections will define the specific constraints of both the categories for our scenario.

4.2.1 Control

We use a temporal logic named as Computation Tree Logic (CTL)[12] to describe the goal and the behavioral constraints. A CTL formula ϕ is described over a set of atomic propositions AP as follows:

$$\phi \rightarrow AP|\neg\phi|true|\phi \wedge \varphi|\phi \vee \varphi|EX\phi|AX\phi|E(\phi U\varphi)|A(\phi U\varphi)|EF\phi|AF\phi$$

The semantics of a CTL formula, denoted by $[[\phi]]$, is given in terms of the sets of states where the formula is satisfied.

- AP is satisfied in all states which are labeled with the propositions in AP.
- $\neg\phi$ is satisfied in states which do not satisfy ϕ.
- $true$ is satisfied in all states.
- $\phi \wedge \varphi$ (resp. $\phi \vee \varphi$) is satisfied in states which satisfy both ϕ and φ (resp. ϕ or φ).
- $EF\phi$ (resp. $AF\phi$) is satisfied in states from which there exists a path (resp. all paths) eventually end in a state satisfying ϕ.

- $E(\phi U \varphi)$ (resp. $A(\phi U \varphi)$) is satisfied in states from which there exists a path (resp. all paths) to a state satisfying φ along which ϕ is satisfied in all states.
- $EX\phi$ (resp. $AX\phi$) is satisfied in states from where at least one of the (resp. all of the) next states satisfy ϕ.

The goal of our scenario is said to be achieved when the Patient service receives a care-taker's Id, in case of an emergency. This will automatically ensure that all the previous messages are successfully communicated. The CTL property of this goal can be written as:

$$A(t_5 \wedge (EF(t_7 \wedge AX(t_0))))$$

Recall from section 4.1 that the parallel composition contains all the possible connections between the services. However, a particular sequence of messages on those connections is required to achieve the desired compositional behavior. We define that sequence with the help of CTL properties as well. For example, the Patient service is connected to the Doctor service to send the location information, but as a requirement, the choreographer should only take this path after the MonitorAndRecorder service has signaled a critical status. In CTL terms:

$$A(\neg t_5 U(t_{12} \wedge AX(t_8)))$$

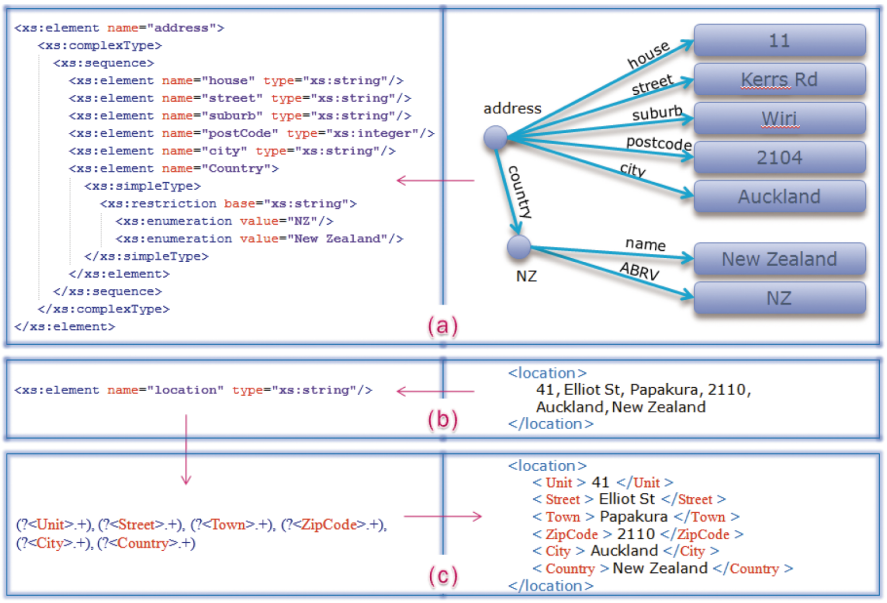

Figure 7. (a) **address** field of the **Doctor** service - *Left*: WSDL extract, *R*ight: Semantic Web from Fig. 4, (b) **location** field of the **Patient** service - *Left*: WSDL extract, *Right*: XML from Fig. 4 (c) *Left*: a Regular Expression for mapping the location field in (b) of the Patient service to a structured complex XML type, shown in (c) *Right*

4.2.2 Data

A data mismatch between two services refers to a mismatch between the output field of the sender service and the input field of the receiver service. All the input and

output fields along with other details such as data-types, message format, protocol details and access methods of a web service, are encapsulated as an XML-based WSDL (Web Services Description Language) model[29]. The common XML syntax ensures that the data matching solutions based on WSDL documents will not suffer from **syntactic mismatches**.

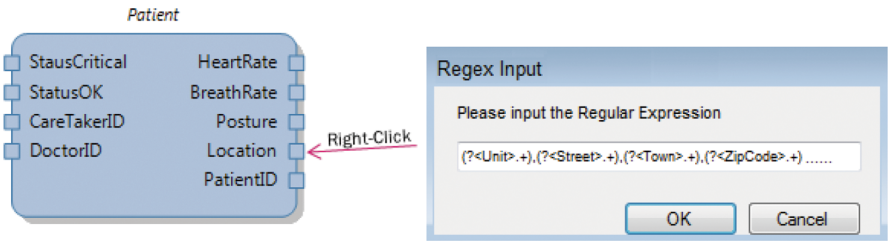

Figure 8. Entering a Regular Expression for mapping the *Location* field of the Patient service

Recall that a **semantic mismatch** occurs when the same piece of data is referred using different names or synonyms by the interacting services. To solve this problem, a universal dictionary has been created that specifies the synonyms, acronyms and abbreviations for the inputs and outputs for the services, as shown in Table 1. The data in the universal dictionary is populated by the help of domain experts and service writers and consumers.

To resolve the **structural mismatch** between a data field of two interacting services, first the input field of the subscriber (receiving) service and the output field of the publisher (sending) service are obtained from their respective WSDL files. Then the order of the sub-fields of the subscriber service is specified. After that the sub-fields are checked for the semantic mismatches using the Universal Dictionary. Finally, sub-fields of the publisher service are traversed to find out if they could form a combination according to the specified order of the subscriber service.

Special Cases: The information obtained from WSDL files can significantly help in a data matching solution, provided all the participating web services fully describe the data fields in their WSDL files, i.e. the complex types are broken down into simple (primitive) XML types. However in practice, web services usually do not simplify the data types unless they need to process the data at the primitive level. For example, Fig. 7(a) shows the well-formed and complex location type from the WSDL model of the Doctor service because a primitive level (unit, street, postCode etc) matching is required to find a hospital with an available doctor. However the Patient service never requires to process locations, and hence just sends out the patient's location without breaking it down into pieces, therefore its location field is represented as a simple string, as shown in Fig. 7(b). This is a special case where the data fields can not be automatically matched, and we need to transform the data manually, or semi-automatically to enable the data communication. A straight forward data transformation solution would be asking the service vendors to re-write the data types by defining the complex types as a set of XML primitive types. However, modifying the existing services is neither a trivial task, nor does it conform with the ideal of composing the *existing* services, thus it should not be used as a solution.

An alternative semi-automatic approach, proposed in this paper, is to employ **regular expressions** for the representation of complex types. We propose an optional feature for the users to enter a regular expression to map any of the input or output ports of the service model, as shown in Fig. 8. The idea is to map the non-granular data fields only. Regular data types can still be obtained from WSDL files as normal. This way, the existing code of the web services need not be altered, and the above solution for structural matching can now be applied to the new, granulated field. Fig. 7(c) *Left* shows a regular expression for the location field of the Vehicle service that maps the string given in Fig. 7(b) in to a granulated complex XML type, shown in Fig. 7(c) *Right*. It should be noted that this data transformation via regular expressions is given as an **optional feature** in our solution.

Table 1 The Universal Dictionary - A lexical database for semantic match-making

Context	Synonyms, Acronyms and Abbreviations		
Address	address	location	destination
	unit	house	apartment
	state	province	...
Person	CCN	creditCardNumber	credit_card_number
	surName	familyName	lastName

5 Composition Framework

5.1 Problem statement

The inputs to the service composition framework are:

Ψ : A set of Computation Tree Logic (CTL) properties that formally describe the behavioral constraints as temporal properties of the composition.

W : A set of services, where every service is specified in WSDL, from which a corresponding SKS has been extracted.

The objective is to solve the following problem:

> Does there exist a choreographer path (C) in the parallel composition (PC) of the given set of services (W), and a set of CTL properties (Ψ) such that all the properties contained in (Ψ) will be satisfied to the choreographer path (C).

5.2 Algorithm

The procedure is detailed in Algorithm 1. It starts by creating a synchronized product or parallel composition to expose all possible communication paths among the web services present in the system. This is done by identifying all the interacting pairs of services (sending and receiving services are termed as publisher and subscriber respectively), and then applying data checks on the fields between the services in

Algorithm 1 Choreographer Synthesis and Verification

```
1: procedure ChorAndVerif(W, Ψ)
2:     PC := ∅
3:     for all PubServ ∈ W do                ▷ Taking each service as a Publisher
4:         for all SubServ ∈ W do                    ▷ All others as Subscribers
5:             if PubServ = SubServ then
6:                 continue
7:             end if
8:             for all o ∈ PubServ.O do          ▷ Set of outputs of the Publisher
9:                 for all i ∈ SubServ.I do      ▷ Set of inputs of the Subscriber
10:                    if DataDictionary(o.name, i.name) then ▷ i/o names matched
11:                        G := getDirectedGraph(i)
12:                        CG := getConnectedGraph(o)
13:                        if G is a subgraph of CG then        ▷ sub-fields matched
14:                            PC.Add(o --σ--> i)       ▷ Add this transition to the PC
15:                        end if
16:                    else
17:                        continue                 ▷ Proceed to the next pair of i/o
18:                    end if
19:                end for
20:            end for
21:        end for
22:    end for
23:    Chor := ∅
24:    Chor := TableauConstructor(PC, Ψ)
25:    if Chor = ∅ then                          ▷ Desired composition not found
26:        return Failure
27:    end if
28:    return Chor
29: end procedure
```

each pair. The names of the data fields, with the help of the universal dictionary, are first checked for the semantic mismatches and simple type conversions like postCode (string) to zipCode (integer). Once the variable names are semantically matched, the next step is to check the structures of the pair. This is done by constructing a directed graph for the input field of the subscriber, and a connected graph (where every node is connected to all other nodes, and can be traversed in any order) for the output field of the publisher. The primitive fields of a complex XML type serve as the nodes of the directed graph while the sequence of the primitives specified in WSDL file defines the direction in graph. The semantic conflicts among the graph's nodes are resolved first using the universal dictionary, then simple graph theory is applied to find out whether the subscriber's graph is a sub-graph of the publisher's (if a graph G' can be constructed by removing edges and/or nodes from graph G then we call G' as a sub-graph of G). The graphs, representing locations for the CareTaker service and the Patient service, are shown in Fig. 9(a) and (b) respectively. Figure 9(c) depicts the successfully matched structure.

The tableau constructor and verifier[35] is adopted for finding the desired

compositional path with the verification of behavioral constraints. The verifier performs CTL model checking using a tableau-based algorithm[7]. The algorithm begins the exploration of the state space from the first state in the parallel composition. The CTL property to be checked is resolved to obtain current and future commitments. The current commitment has to be fulfilled by the current state while the future commitment is evaluated for the successor states. The algorithm terminates if the complete state-space has been explored or all the commitments are fulfilled. A successfully generated choreographer path for the discussed scenario is presented in Fig. 10. For the execution, each transition in the choreographer path will be translated to the respective service call.

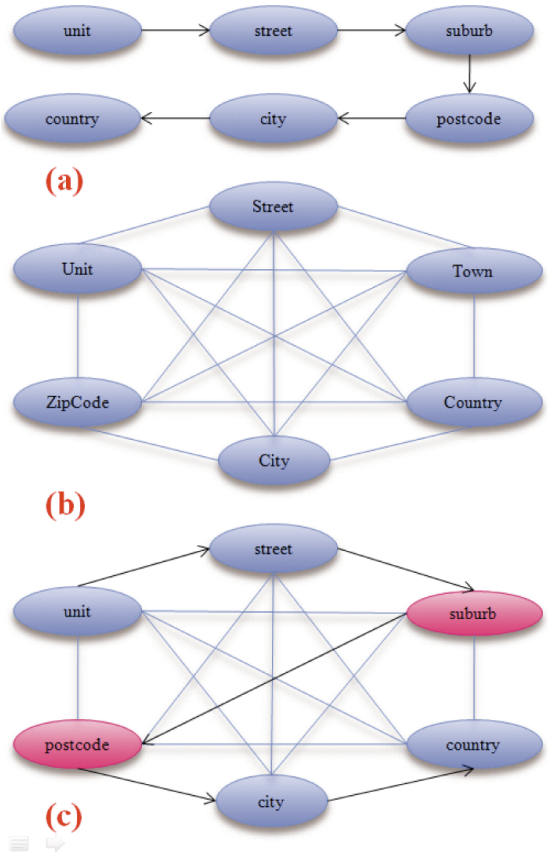

Figure 9. **Data-structures for location data fields**. (a) Directed Graph for the Subscriber (Care Taker) service. (b) Connected Graph that connects all the primitives of the Publisher (Patient) service. (c) Subscriber's Path is found in the Publisher's graph after semantically matching the mismatched fields (nodes)

6 Experimental Results

The technique is evaluated via common use cases. The same use cases, discussed in Ref. [1] are taken, so that the results can be compared. The first example is *HelpMeOut* which is the running example in Ref. [1]. It consists of

several independently existing web/mobile services: Vehicle, Mechanic, Insurance, and Workshop services. The *HelpMeOut* system makes it easier for a vehicle's driver to call for assistance in case of an emergency. The second example is a travel reservation service, modified from Ref. [14], where the aim is to develop a composite service (a) for reserving transportation (car-rental), (b) for buying the airline tickets, (c) for booking accommodation (hotel), (d) for credit card payments and (e) for customer's details and preferences. The composition is realized from existing services, one each for car-rental, air-ticket purchase, hotel reservation, payment and customer. The third example is regarding post surgery monitoring of a heart-condition patient, and is the running example of this paper. It involves Patient, Doctor, CareTaker, Emergency and MonitorAndRecorder services. The composition in this case uses individual services to (i) maintain a regular transfer of a patient's vital signals, and to (ii) determine doctor and care-taker availability in-case of an emergency.

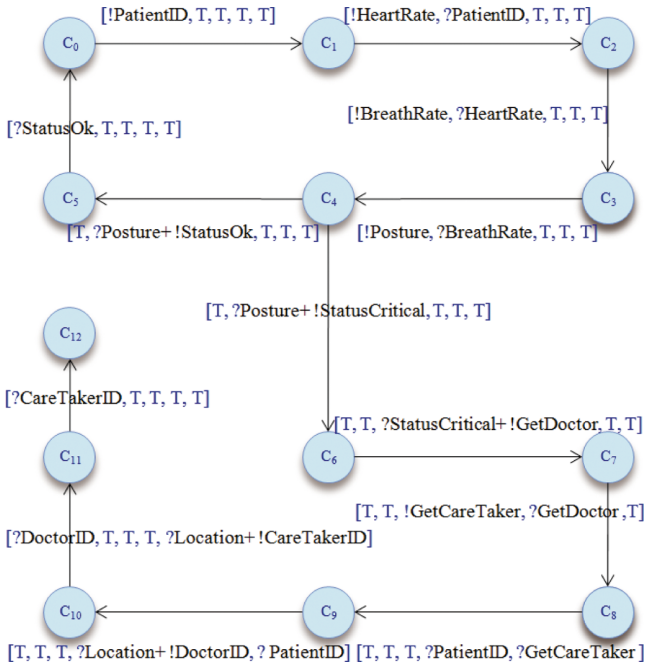

Figure 10. States and Transitions in the Choreographer path. *?* shows the input events while *!* depicts the output events

For each of the above examples, Table 2 presents the number of services involved in the composition, the number of states, transitions along with the time taken to generate the choreographer in the proposed approach as well as in Ref. [1]. Table 2 also presents the formats of the independent services and the data mismatches resolved during compositions; where db, csv, s. web and xls stand for database, comma-separated-values, semantic-web and excel spreadsheet respectively. 100% data matching indicates successful composition for the use-cases *HelpMeOut* and *MyHeartCare*, while 85% of data matching for the use-case *Travel Reservation* shows that the data structures of some of the fields could not be matched, and the

mismatched services are required to be replaced for a successful composition.

The difference between the current approach and the previous one ([1]) can be determined by noticing the number of states and transitions generated for the choreographers in each case. It can be seen that the choreographers are identical for the cases (*HelpMeOut* and *MyHeartCare*) where all the data mismatches are successfully resolved. However, for the case (*Travel Reservation*) where the data mismatches could not be resolved, the proposed approach returns an immediate failure; while the previous approach still generates a choreographer (that's behaviorally correct only), and then checks for a data mismatch within the choreographer path which eventually returns a failure. This difference is also reflected in the time taken for choreographer synthesis, especially for the failure cases where the time difference is almost the half of the other. For the successful compositions, the current algorithm also performs slightly better in terms of time-taken, as shown in the results.

Table 2 Results - Identical experiments are performed to compare with the results of Ref. [1]

	No. of	C [proposed]			$C^{[1]}$		
Use Case	Services	States	Tran	Time(s)	States	Tran	Time(s)
HelpMeOut[1]	4	8	9	1.25	8	9	1.41
Travel Res.[14]	5	Not found		0.67	18	19	1.19
MyHeartCare[39]	5	12	13	1.51	12	13	1.72

	Service Data Format					Data Matches(%)		
Use Case	1	2	3	4	5	semantic	structural	syntactic
HelpMeOut[1]	xml	xml	csv	s. web	-	100	100	100
Travel Res.[14]	db	db	db	db	db	100	85	85
MyHeartCare [39]	csv	xml	xls	db	db	100	100	100

7 Conclusions

In this paper we presented a formal approach, based on model checking, for web service composition. The objectives are to integrate independently existing web services to get the desired compositional behavior, and to resolve the data mismatches among the participating services at different levels. These objectives have been achieved using a tableau based algorithm to realize compositions in a goal-directed fashion, and using WSDL and regular expressions to generate data models to detect and resolve data mismatches. Experiments show that the technique is promising and can be applied practically. Currently, we are working on multiple extensions of the current approach - mainly on the automatic generation of a finite state machine for a web service from its WSDL document. Secondly, we are looking for a composition solution to address functional requirements, low-level behavioral constraints, and preferences for non-functional properties, all in a single unified framework. Other probable avenues of research include investigating the

feasibility of the approach for other service oriented architectures such as mobile services and cloud computing, and to extend the data matching solution by employing semantic web techniques.

References

[1] Ali S, Roop P, Warren I, Bhatti Z. Unified management of control flow and data mismatches in web service composition. Service Oriented System Engineering (SOSE), 2011 IEEE 6th International Symposium on (dec. 2011). 93 –101.

[2] Alur R, Dill DL. A theory of timed automata. Theor. Comput. Sci. 126 (April 1994). 183–235.

[3] Andrews T, Curbera F, Dholakia H, Goland Y, Klein J, Leymann F, Liu K, Roller D, Smith D, Thatte S, Trickovic I, Weerawarana S. BPEL4WS, Business Process Execution Language for Web Services Version 1.1. IBM. 2003.

[4] Barker A, Walton CD, Robertson D. Choreographing web services. IEEE Transactions on Services Computing 2. 2009. 152–166.

[5] Berardi D, Calvanese D, Giacomo GD, Lenzerini M, Mecella M. Automatic composition of e-services that export their behavior. Proc. 1st Int. Conf. on Service Oriented Computing (ICSOC), volume 2910 of LNCS. Springer. 2003. 43–58.

[6] Berners-Lee T, Hendler J, Lassila O. The semantic web. http://www.scientificamerican.comarticle.cfm?id=the-semantic-web. 20 May 2009. 2001.

[7] Cleaveland R. Tableau-based model checking in the propositional mu-calculus. Acta Inf. 27. August 1990. 725–747.

[8] Creager D, Simpson AC. A fully generic, graph-based approach to data transformation discovery. Proc. of GMC (Graph Computation Models) 2006. 2006.

[9] Creager D, Simpson AC. Towards a fully generic theory of data. Proc. of ICFEM 2006 2006. Springer-Verlag Lecture Notes in Computer Science, volume 4260: 304–323.

[10] Di Pietro I, Pagliarecci F, Spalazzi L. Model checking semantically annotated services. Software Engineering, IEEE Transactions on PP, 2011, 99(1).

[11] Doan A, Madhavan J, Domingos P, Halevy A. Learning to map between ontologies on the semantic web. Proc. of the 11th international conference on World Wide Web. (New York, NY, USA). WWW '02. ACM. 2002. 662–673.

[12] Emerson EA, Halpern JY. Decision procedures and expressiveness in the temporal logic of branching time. J. Comput. Syst. Sci. 30. February 1985. 1–24.

[13] Fu X, Bultan T, Su J. Formal verification of e-services and workflows. CAiSE '02/ WES '02: Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web. London, UK. Springer-Verlag. 2002. 188–202.

[14] Haas H. "web service use case: travel reservation". May 2002.

[15] Jay. Choreography and orchestration : A software perspective. November 2009.

[16] Kavantzas N, Burdett D, Ritzinger G, Fletcher T, Lafon Y, Barreto C. Web Services Choreography Description Language Version 1.0. November 2005.

[17] Kazhamiakin R, Pistore M. Static verification of control and data inweb service compositions. Proc. of the IEEE International Conference on Web Services. IEEE Computer Society. Washington, DC, USA. 2006. 83–90.

[18] Kleinpell R, Avitall B. Integrating telehealth as a strategy for patient management after discharge for cardiac surgery: Results of a pilot study. J Cardiovasc Nurs 22, 2007, 1: 38–42.

[19] Marconi A, Pistore M. Synthesis and composition of web services. In Bernardo M, Padovani L, Zavattaro G, eds. Lecture Notes in Computer Science. SFM, Springer. 2009, 5569: 89–157.

[20] Mcguinness DL, van Harmelen F. OWL web ontology language overview. W3C recommendation. W3C. Feb. 2004.

[21] McIlraith SA, Son TC, Zeng H. Semantic web services. 2001. Intelligent Systems, IEEE.

[22] Milanovic N, Malek M. Current solutions for web service composition. Internet Computing, IEEE 8. (nov.-dec. 2004), 6: 51–59.

[23] Mitra S, Basu S, Kumar R. Local and on-the-fly choreography-based web service composition. Proc. IEEE/WIC/ACM Int Web Intelligence Conf. 2007. 521–527.

[24] Mitra S, Kumar R, Basu S. Automated choreographer synthesis for web services composition using i/o automata. Web Services, 2007. ICWS 2007. IEEE International Conference on. July 2007. 364 –371.

[25] Mitra S, Kumar R, Basu S. Optimum decentralized choreography for web services composition. Services Computing, 2008. SCC '08. IEEE International Conference on. July 2008, 2: 395–402.

[26] Mitra S, Kumar R, Basu S. A framework for optimal decentralized service-choreography. Proc. IEEE Int. Conf. Web Services ICWS 2009. 2009. 493–500.

[27] Mocan A, Cimpian E. D13.3v0.2 wsmx data mediation. 2005.

[28] Moreau A, Malenfant J, Dao M. Data flow repair in web service orchestration at runtime. Proc. of the 2009 Fourth International Conference on Internet and Web Applications and Services. IEEE Computer Society. Washington, DC, USA. 2009. 43–48.

[29] Moreau JJ, Chinnici R, Ryman A, Weerawarana S. Web services description language (WSDL) version 2.0 part 1: Core language. Candidate recommendation, W3C. Mar. 2006.

[30] Oinn T, Addis M, Ferris J, Marvin D, Senger M, Greenwood M, Carver T, Glover K, Pocock MR, Wipat A, Li P. Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20, 2004, 17: 3045–3054.

[31] Ouksel AM, Sheth A. Semantic interoperability in global information systems. SIGMOD Rec. 28. (March 1999). 5–12.

[32] Pathak J, Basu S, Lutz R, Honavar V. Parallel web service composition in MoSCoE: A choreography-based approach. Web Services, 2006. ECOWS '06. 4th European Conference on. Dec. 2006. 3–12.

[33] Rahm E, Bernstein PA. A survey of approaches to automatic schema matching. The VLDB Journal 10. Dec. 2001. 334–350.

[34] Roman D, Keller U, Lausen H, de Bruijn J, Lara R, Stollberg M, Polleres A, Feier C, Bussler C, Fensel D. Web service modeling ontology. Applied Ontology 1, 2005, 1: 77–106.

[35] Sinha R. Automated Techniques for Formal Verification of SoCs. [PhD thesis]. The University of Auckland. Feb 2009.

[36] Sinha R, Roop P, Basu S, Salcic Z. Multi-clock SoC design using protocol conversion. Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09. April 2009. 123–128.

[37] ter Beek M, Bucchiarone A, Gnesi S. Web service composition approaches: From industrial standards to formal methods. Proc. of the Second International Conference on Internet and Web Applications and Services. IEEE Computer Society. Washington, DC, USA. 2007. 15.

[38] Ter Beek MH, Ellis CA, Kleijn J, Rozenberg G. Synchronizations in team automata for groupware systems. Comput. Supported Coop. Work 12. Feb. 2003. 21–69.

[39] Warren I, Weerasinghe T, Maddison R, Wang, Y. Odintelehealth: A mobile service platform for telehealth. Procedia Computer Science 5. 2011. 681–688.