Automated Service Composition Via Supervisory Control Theory*

Francis Atampore, Juergen Dingel, Karen Rudie

Abstract—Web services play a major role in electronic businesses and allow organizations to perform certain business activities in a distributed fashion. In some circumstances, a single service is not able to perform certain tasks and it becomes imperative to compose two or more services in order to complete a task. While approaches to tackle such a problem are known, the task of generating provably correct Web service compositions still remain challenging and complex. In this paper, we develop a supervisory control framework for automated composition of Web services. Labelled Transition Systems augmented with guards and data variables are used to represent a given set of Web service specifications. We model the interactions of services asynchronously and we use guards and data variables to allow us to express certain preconditions which are then propagated from the system requirements through the overall composite service. The objective of our framework is to synthesize a controller, which interacts with a given set of Web services through messages to guarantee that a given specification is satisfied. A key novelty of this work is the application of control theory to service-oriented computing and the incorporation of run-time input into the supervisor generation process.

I. INTRODUCTION

Today, we are undergoing a major paradigm shift in software development since the emergence of the Web. The vision underpinning Web services can be fully achieved if we envision a collaboration between a community of numerous service providers and service consumers who interact in order to achieve certain business goals. Thus, one of the key functionalities of Web services is service composition, which seeks to create, select and integrate pre-existing services to develop new value-added services and applications. Hence, it promotes the rapid development of software systems by reducing the cost and effort for developing new services from scratch and also promotes re-usability. A number of approaches have been proposed to tackle this problem and most of these existing approaches are motivated by the works in AI planning and cross-enterprise workflow [1], [2]. However, the task of generating provably correct Web service compositions still remains challenging and complex.

In this paper, we develop a formal framework, Supervisor Aware Service Composition Architecture (SASCA), for modeling Web service composition based on Supervisory Control Theory (SCT) of Discrete-Event Systems (DES). The classical Ramadge-Wonham approach [3] to the supervisory control problem (SCP) is defined as follows: given a plant \( G \) modeled in the form of a state-transition system which captures the behaviours of the plant according to some possible events, given a set of specifications \( L \) which describes the legal sequences of events of the plant, synthesize a supervisor \( S \) so that \( S \) restricts \( G \) in such a way that all its executions satisfy \( L \) and such that \( S \) is minimally restrictive. The DES control methods fit well into the problem of automatic service composition if the problem is reduced to observing events of a system and restricting its behaviour to specific sequences. Hence, we can apply existing DES techniques and algorithms to address the problem of automatic service composition. Supervisory control synthesis has several benefits which are as follows. It results in a correct-by-construction control synthesis, and the generated controller is minimally restrictive by preventing a system behaviour only if it violates the system requirements. It also relies on automata theory to provide a well-defined syntax and semantics for modeling systems which could be very useful for specifying services. Supervisory control theory has been applied to software systems such as concurrency in multithreaded programs and component based software systems [4]. Here we apply SCT to address the problem of Web service composition.

A. Overview of Approach

The approach we propose automatically derives Labelled Transition Systems (LTSs) equipped with guards, and data variables from a given set of Web services specification in WS-BPEL [5]. To this end, we provide an SCT modeling formalism for LTSs and then we describe a novel technique to synthesize a composition satisfying a given functional requirement (data and control flow) also specified in LTS. In this way, the problem of orchestrating data and control flow requirements can be achieved by using the notion of controllability in DES where the supervisor enacts control by disabling and enabling certain actions in order to enforce the given goal. The inputs to the system are the set of Web services specified in WS-BPEL and the requirements are also specified as LTSs. Internally, we represent the given Web services as LTSs and a number of preprocessing and refinement steps are performed to make the available services suitable for composition. Next, the proposed supervisory control framework based on LTSs is applied to synthesize a composition and finally, we transform the resulting LTS into a WS-BPEL executable file as an output. Figure 1 shows the basic architectural diagram for the SASCA framework. Our approach lies between the event-based supervisory control [3] and state-avoidance control problem [6]. That is, apart from the supervisor being able to prevent certain events from occurring by properly disabling and enabling controllable events, the supervisor is also able to prevent the system from reaching certain sets of states designated as forbidden states by using run-time information of variables in the system. This is due to the fact that some of the events in our model are block-box (atomic actions) in nature which exhibit non-deterministic properties. We handle non-determinism through model refinement and the adaptation of event enforcement
Fig. 1: Supervisor Aware Service Composition Architecture (SASCA)

supervisory control theory [7]. Therefore, the generated supervisor not only restricts the behaviour of the plant, but also has the ability to actively enforce certain events. In addition, the generated supervisor is able to restrict the system by assigning stronger guards to data variables. This allows us to control the data a service can send or receive. The use of run-time information in our model is inspired by the fact that the services we model are non-deterministic and partially controllable. That is, the outputs of a service cannot be predicted a priori and some of the internal computations of a service are hidden from other external services. For example, whether there are still seats available on an airline flight cannot be known until run-time.

We propose a novel supervisory control framework for automated composition of Web services. We provide some insights that allow users to express and define functional requirements (data and control flow) for a composition. We formalize the problem of automated composition of services and provide an SCT formalism for LTSs that captures the semantics of Web services and the theoretical basis underlying our framework. We then develop a set of algorithms to generate a composition (controller) satisfying a given functional requirement also specified in LTSs. Beyond the standard disabling and enabling of events, the generated controller in our framework has the ability to enforce certain events based on run-time information to actively drive the system towards its goal.

B. Service Composition Problem and Motivating Example

In this section, we present the composition problem by means of a classical example commonly used to illustrate the problem Web service composition. Informally, the service composition problem comes into play when a request to a service is not satisfied by a single service (a component service) and then multiple services are identified and composed into a new service (composite service) to satisfy the request by orchestrating the services involved. Let us consider a Flight Reservation and Purchase System (FRPS) that offers customers travel packages by allowing customers to make a reservation for a specified airline and make payment in order to reserve the flight. All interactions are managed by Web services. The objective of the FRPS can be attained by composing an Airline Service, a Bank Service, a Hotel Service and an On-line Customer Interface Service. These services are represented in WS-BPEL. The main challenge is how to compose these services so that the user can directly ask the combined service to reserve and purchase a ticket satisfying some given system requirements. In the following, we will provide an informal description of these services.

- The Flight service is designed to receive requests for booking a specified flight for a given date and location. It checks an internal database for flight availability, sends an offer with a cost and a flight schedule in response to the client’s request. The client can either accept or refuse the offer, if the client decides to accept the offer, the FRPS will book the flight and provide additional information such as an electronic ticket.
- The Bank Service receives requests to check that a credit card, debit card or money order can be used to make a payment and provides an option for a client to check current balance of an account or withdraw from the account to make payment for a purchase. The transaction may fail if the card provided is not valid or if there are not sufficient funds in the client’s accounts.
- The Hotel Service accepts requests to provide information on available hotels for a given date and a given location. It checks for the availability of hotels and selects a specific hotel based on the client’s request and returns an offer with a cost and other hotel information.
- On-line User Interface Service serves as a customer interface through which the client can interact with the FRPS; it receives input messages from the user and sends output messages to the user as well as to facilitate interactions among the available services.

II. SERVICE AND SCT REPRESENTATION

In this paper, we assume that each WS-BPEL of a Web service is formally represented by what we call a Service Labelled Transition System (SLTS). The evolution of the
system from one state to another is determined by its current state and the evaluation of a guard of a transition.

**Definition 2.1: A Service Labelled Transition System** is modeled by a tuple \( G_W = (S, S^0, I, O, A, \Gamma, S^F) \) where

- \( S \) is a finite set of states
- \( S^0 \subseteq S \) is the set of initial states
- \( \Sigma = I \cup O \cup A \cup \{ \tau \} \) is the set of events, where \( I, O, A \) denote the set of input messages \((\text{?}m(t))\), output messages \((m(t))\) and atomic operations, respectively, and \( I \cap O = \emptyset \), and \( \tau \) represents an internal unobservable event
- \( V = \{ v_1, ..., v_n \} \) is a finite set of data variables over a given domain
- \( \Gamma \subseteq S \times (I \cup O \cup A) \times G \times S \) is the transition relation, where \( G \) is the set of guards \( g \) over a subset of the variables in \( V \)
- \( S^F \subseteq S \) is the set of final states

We employ infix notation and we write \( s \xrightarrow{\alpha} s' \) as shorthand for \( (s, g, e, s') \in \Gamma \). A tuple \( \lambda \in \Gamma \), \( \lambda = s \xrightarrow{\alpha} s' \) is a transition in \( G_W \), where \( e \in \Sigma \), and \( g \) is a guard in \( G \), a condition or a predicate defined over variables and formulas. The absence of an explicit guard on a transition means that the condition is always true. The dynamics of an SLTS depend on the current state of the system and on the valuation of the transition guards with respect to the current value of a variable. In the sequel, let \( \Sigma^* \) denote the set of all finite strings of the form \( \alpha_1 \alpha_2 ... \alpha_n \) of events from \( \Sigma \), including the empty string \( \epsilon \).

The possible behaviour of an SLTS is modeled by the set of executions. The execution of an SLTS is represented by the set of all possible runs. A run of an SLTS is a sequence of \( r = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} ... \xrightarrow{\alpha_{n-1}} s_n \) such that \( \forall i < n, s_i \xrightarrow{\alpha_i} s_{i+1} \in \Gamma \) and the trace of the run is given by \( \alpha_1 \alpha_2 ... \alpha_{n-1} \). The language generated by an SLTS, denoted by \( L(G_W) \), is the set of words \( L(G_W) = \{ w \in \Sigma^* | s_0 \xrightarrow{w} s, s_0 \in S^0, s \in S^F \} \) where \( s_0 \xrightarrow{w} s \) denotes a multi-step transition relation which is defined inductively as a finite sequence of application of a transition relation which produces a state \( s \) that a sequence of events leads to from the initial state \( s_0 \). The formal language that we define here differs from extended finite state automata [8] in that we do not require an update function. Hence, variables can only be updated through the atomic operation.

We distinguish between two kinds of transitions, the first one is a static transition \( s \xrightarrow{\alpha} s' \) which does not depend on a variable; the guard of this transition is always true and is triggered when the event on the transition takes place. The second type of transition is a dynamic transition \( s \xrightarrow{\alpha} s' \) which have guards that depend on a variable which is fired only if the guard on the transition evaluates to true and the event is enabled. For example, in Figure 2(a), the transition \( S_1 \xrightarrow{\text{if}\text{Request(date,loc)}} S_2 \) is static whereas the transition \( S_3 \xrightarrow{\text{process Booking}()} S_4 \) is dynamic.

**Definition 2.2: (Subguards)** Let \( g_1 \) and \( g_2 \) be two guards. We call \( g_2 \) a subguard of \( g_1 \) denoted by \( g_2 \preceq g_1 \), if \( g_2 \) is stronger than \( g_1 \), i.e., \( g_1 \land g_2 = g_2 \).

In order to model behaviours common to two or more SLTSs, we define parallel product in such a way that an event can be executed only if it is contained in all the SLTSs involved. This will allow us to model multiple requirements of a system. For example, given \( SLTS_1 \) and \( SLTS_2 \) specifying some system requirements, an event is allowed to occur only if it is allowed in both SLTSs.

**Definition 2.3: (Parallel Product)** Given two SLTSs \( G_{W_1} = (S_1, S^0_1, I_1, O_1, A_1, \Gamma_1, S^F_1) \) and \( G_{W_2} = (S_2, S^0_2, I_2, O_2, A_2, \Gamma_2, S^F_2) \) their parallel product is given by \( G_{W_1} \times G_{W_2} = (S_1 \times S_2, S^0_1 \times S^0_2, I_1 \cup I_2, O_1 \cup O_2, A_1 \cup A_2, \Gamma_1 \times \Gamma_2, S^F_1 \times S^F_2) \) such that the transition relation \( \Gamma_1 \times \Gamma_2 \) is defined as follows.

- \( s_1, s_2 \xrightarrow{\alpha_1 \alpha_2} s'_1, s'_2 \in \Gamma_1 \times \Gamma_2, \alpha \in \Sigma_1 \cap \Sigma_2 \) if \( s_1 \xrightarrow{\alpha_1} s'_1 \in \Gamma_1 \) and \( s_2 \xrightarrow{\alpha_2} s'_2 \in \Gamma_2 \).
- Undefined otherwise

In the parallel product, the transitions of two SLTSs must always synchronize on shared events \( \Sigma_1 \cap \Sigma_2 \). Where \( \Sigma_1 = (I_1 \cup O_1 \cup A_1) \) and \( \Sigma_2 = (I_2 \cup O_2 \cup A_2) \). Analogous to the standard SCT where the legal language is a sublanguage of the plant, in our framework we use the notion of simulation relation to describe the relationship between a system and a given specification, both modeled as SLTSs.

**Definition 2.4: (Simulation Relation with Guards)** Given two SLTSs \( G_{W_1} = (S_1, S^0_1, I_1, O_1, A_1, \Gamma_1, S^F_1) \) and \( G_{W_2} = (S_2, S^0_2, I_2, O_2, A_2, \Gamma_2, S^F_2) \), \( G_{W_1} \) simulates \( G_{W_2} \), denoted by \( G_{W_1} \preceq G_{W_2} \), if there exists a relation \( R \subseteq S_1 \times S_2 \) such that \( \forall (s_1, s_2) \in R, \text{if } s_1 \xrightarrow{\alpha_1 \alpha_2} s'_1 \in \Gamma_1 \text{ then } \exists s_2', g_2 \text{ such that } (s_2 \xrightarrow{\alpha_2} s'_2 \in \Gamma_2, g_1 \preceq g_2) \) and \( (s'_1, s'_2) \in R \).

That is, every transition taken by \( G_{W_1} \) can be matched by \( G_{W_2} \). In essence, when the two SLTSs are represented in their respective execution tree (i.e., unfolding a given LTSs possibly infinite) \( G_{W_1} \preceq G_{W_2} \) means that \( G_{W_2} \) has a larger execution tree than \( G_{W_1} \).

**A. Service Representation**

In our framework, we assume that Web services are described in WS-BPEL; based on this we automatically extract the SLTS models. Formally, we model a service as an SLTS as defined above. **Input and Output Messages:** we denote a reception of a message as \( \text{?}m(t) \) and an emission of message as \( m(t) \), where \( m \) is the name of the message and \( t \) is the set of data parameters or variables.

Variables are local to a service and a variable cannot be modified by more than one service at a given time. **Atomic Operations:** operations such as function invocations are denoted by \( \text{nameOperation}(I :: O) \) with input parameters \( I \) and output parameters \( O \). The effects that the atomic operation has on its output variables are only visible inside the service invoking the operation. The output of an atomic operation is unknown until runtime. Figure 2 shows the SLTS representations of three of the component services of our FRPS example. The customer interface component service in omitted here due to limited space.
III. Supervisor Aware Service Composition Architecture (SASCA)

A. Service Composition Synthesis (Supervisor Synthesis)

In this section, based on the representation of services using SLTSS, we formalize the problem of composing Web services. Our model of synthesized Web services relies deeply on message passing, interaction with data and action. The composition problem (formalized in Definition 3.7 below) that we consider here is as follows: given a set of available services $\mathcal{G}_W_1, \mathcal{G}_W_2, ..., \mathcal{G}_W_n$ and a set of specifications $T^W$ representing the goal (or desired) service over the same environment (same set of atomic actions), we would like to construct a controller $C$ satisfying some controllability and nonblocking constraints which interacts with the available services to satisfy the specification $T^W$. Thus, $C$ serves as a controller that restricts the system in such a way that all its executions satisfy $T^W$ and that $C$ is minimally restrictive. In addition to requiring that the generated controller satisfies the controllability and nonblocking criteria, the controlled system is also free of errors that may result from communication among component services. We assume that both the available services and the goal service are expressed in SLTSS as defined above.

B. Asynchronous Communication

In our formalism, we use asynchronous communication to model the interaction among the available services. Asynchrony can be achieved by employing unbounded memory to store the variables and parameters set from one component service. However, in this work the way we model service interactions does not make use of any particular medium for storing and retrieving messages.

As a first step in the composition process we obtain $\mathcal{G}_W$ by combining the set of available services whose SLTSS is given by $\mathcal{G}_W_1, \mathcal{G}_W_2, ..., \mathcal{G}_W_n$ by means of asynchronous parallel composition as defined below.

**Definition 3.1: (Asynchronous Parallel Composition)**

Given two SLTSS $\mathcal{G}_W_1 = (S_1, S^0_1, I_1, O_1, A_1, \Gamma_1, S^F_1)$ and $\mathcal{G}_W_2 = (S_2, S^0_2, I_2, O_2, A_2, \Gamma_2, S^F_2)$ their asynchronous parallel composition is given by $\mathcal{G}_{W_1 || W_2} = (S_1 \times S_2, S^0_1 \times S^0_2, I_1 \cup I_2, O_1 \cup O_2, A_1 \cup A_2, \Gamma_1 \parallel \Gamma_2, S^F_1 \times S^F_2)$ such that the transition relation $\Gamma_1 \parallel \Gamma_2$ is defined as follows:

- $(s_1, s_2) \xrightarrow{\alpha_{[g_1]}} (s'_1, s'_2) \in \Gamma_1 || \Gamma_2$, if $s_1 \xrightarrow{\alpha_{[g_1]}} s'_1 \in \Gamma_1$
- $(s_1, s_2) \xrightarrow{\alpha_{[g_2]}} (s'_1, s'_2) \in \Gamma_1 || \Gamma_2$, if $s_2 \xrightarrow{\alpha_{[g_2]}} s'_2 \in \Gamma_2$
- Undefined otherwise

Definition 3.1 can be extended to $n$ number of services by observing that it is associative, i.e., $(\mathcal{G}_W_1 || (\mathcal{G}_W_2 || \mathcal{G}_W_3)) = (\mathcal{G}_W_1 || \mathcal{G}_W_2) || \mathcal{G}_W_3)$. Therefore, without ambiguity we can write $\mathcal{G}_W_1 || \mathcal{G}_W_2 || \mathcal{G}_W_3 || \mathcal{G}_W_n$ to represent the composition of multiple Web services.

Definition 3.1 allows individual services to make independent moves and describes all possible behaviours of a given set of available services. We assume that the available services do not interact among themselves; any form of communication is through the supervisor. Hence, we require that the input (output) messages of a service are disjoint from the output (input) of another service. The definition of synchronous parallel composition given above is defined so that the output from one service is consumed by the input of another service to produce an internal action such that the guards on both the output and input transitions are satisfied.

C. Preprocessing Design Errors

In general, modeling the composition of asynchronous communicating systems could result in various kinds of undesirable behaviours such as unspecified receptions and non-executable interaction of the system [9]. Given the system to be controlled which is represented by the asynchronous parallel composition of available services $\mathcal{G}_W = \mathcal{G}_W_1 || \mathcal{G}_W_2 || \mathcal{G}_W_n$, $\mathcal{G}_W$ may contain the following errors as defined below. We want the combined set of services to be free from unspecified receptions and non-executable
interactions. An unspecified reception is a situation where one service can send a message at a reachable state, but other services are not able to receive it. That is, the SLTS description of a service contains an emission that cannot be consumed by the related component services involved in the composition. Similarly, non-executable interactions refer to a situation in which one service is able to receive a message that has not already been sent by some other service. This results in additional unmatched receptions. An SLTS \( G_W = (S, S^0, I, O, A, \Gamma, S^F) \) is said to be communication-error free if it is free from unspecified receptions and non-executable interactions. Given the SLTS \( G_W \) representing the asynchronous parallel product of the available services, we perform a refinement or preprocessing on \( G_W \) to get a communication-error free SLTS. This preprocessing step removes all paths that contain any unspecified and non-executable interaction errors from the original plant.

D. Requirements

The composition requirements are also given as SLTSs which specify the possible accepted interactions that must hold in the composition (plant). In this framework, specification can take one of the following forms: (i) A composition requirement can specify a set of constraints on the ordering of events and actions. A typical example of this constraint in our flight booking system is that the credit card of the user must be verified by the bank service before a booking confirmation is delivered to the customer. (ii) Another form of composition requirement is to specify stronger guards that limit the values that can be taken by a variable or a data parameter from a given domain. This can be used to restrict the values of a variable that can be sent or received by services. In Figure 3(a), the SLTS specifies that the airline service from our running example can only accept reservations for KLM and Delta (\((av == KLM) \lor (av == Delta)\)) but not Air Canada. This specification restricts the values of the variable \(av\). Hence, a correct composition must not allow Air Canada reservations to be made. Figure 3(b) also specifies the kind of payment that can be made by a client to the Flight Reservation and Purchase System. The SLTS specifies that the system can only accept payment by credit card or debit card. (iii) One can also explicitly specify a set of forbidden states that the system should not reach during execution. For example, a specification that specifies that the cost of a product \(c\) should not exceed a limit \(m\), i.e., \(c < m\) implies that \(c \geq m\) leads to an unsafe state. There could be multiple specifications. In that case, we use parallel product to put them together. Intuitively, a supervisor in this case will be one that guarantees that all specifications are achieved.

E. Controller Synthesis

In this section, we study how to synthesize a controller that will ensure that the systems behaviour satisfies the given requirements. We assume that the system to be controlled is given by the asynchronous parallel product of the available services \( G_{W_1} \parallel G_{W_2} \parallel \ldots \parallel G_{W_n} \) and the system requirements (target service) are given by \( T^W \). Now, we require that the asynchronous product of the available services simulates the goal services. That is, \( T^W \preceq G_W \). In the case that it does not simulate the goal service we perform refinement on the target services.

The following definition specifies the parallel product with refinement to safe and forbidden states.

Definition 3.2: (Refinement Composition) Given two SLTSs \( G_{W_i} = (S_i, S^0_i, I_i, O_i, A_i, \Gamma_i, S^F_i) \) and \( T^W = (S_2, S^0_2, I_2, O_2, A_2, \Gamma_2, S^F_2) \) representing the plant and the specification respectively, and let \( s_f \) denote a bad or forbidden state, we can compute their parallel product as well as refine the plant with respect to the specification in such a way that the behaviour not allowed by the specification results in bad or forbidden states in the plant. The composition refinement of the plant and the specification denoted by \( G_{W_1} \times ref T^W \) is given by \( R = (S_1 \times (S_2 \cup \{s_f\}), S^0_1 \times S^0_2, I_1 \cup I_2, O_1 \cup O_2, A_1 \cup A_2, \Gamma_1 \times \Gamma_2, S^F_1 \times S^F_2) \), and the transition relation \( \Gamma_1 \times \Gamma_2 \) is defined as follows.

- \((s_1, s_2) \xrightarrow{\alpha[g_1 \land g_2]} (s'_1, s'_2) \in \Gamma_1 \times \Gamma_2 \) and \((s_1, s_2) \xrightarrow{\alpha[g_1 \lor g_2]} (s'_1, s_f) \in \Gamma_1 \times \Gamma_2 \), if \( s_1 \xrightarrow{\alpha[s_1]} s'_1 \in \Gamma_1 \) and \( s_2 \xrightarrow{\alpha[s_2]} s'_2 \in \Gamma_2 \).
- \((s_1, s_2) \xrightarrow{\alpha[s_2]} (s'_1, s_f) \in \Gamma_1 \times \Gamma_2 \), if \( s_1 \xrightarrow{\alpha[s_1]} s'_1 \in \Gamma_1 \) and \( s_2 \xrightarrow{\alpha[s_2]} s'_2 \notin \Gamma_2 \).
- Undefined otherwise

The set of states of \( R \) is given by \( Y = S_1 \times (S_2 \cup \{s_f\}) \). A state \((s_1, s_2) \in Y \) is said to be forbidden if \( s_2 = s_f \). States that are not in \( \{s_f\} \) are called safe states. Now, if all forbidden states in \( R \) are not reachable, we will call \( R \) a safe SLTS.

We assume that the set of events \( \Sigma \) is partitioned into three subsets namely, controllable events \( \Sigma_c \subseteq \Sigma \), uncontrollable events \( \Sigma_u \subseteq \Sigma \), and forcible events \( \Sigma_f \subseteq \Sigma \) such that controllable events can be disabled by the controller while uncontrollable events cannot be prevented from occurring. In addition, the forcible events are special events that can be enforced by the controller. They are able to preempt both controllable and uncontrollable events at run-time but not static transitions. We do not assume any relationship between the set of controllable and enforceable events at this moment. The notion or the intent of control in this framework involves
the following techniques. Firstly, the generated controller prevents the system from taking events or transitions that violate the control requirement and secondly, it also prevents the system from reaching states designated as forbidden. In order to achieve the above control goals the supervisor enacts control based on the following three control criteria.

1) Disabling of controllable events on a transition
2) Assignment of stronger guards to controllable transitions
3) Enforcement of enforceable events.

To develop our control synthesis algorithms and strategies, we assume that the system evolves from one state to another based on the kind of transitions (static or dynamic) at a given state. Thus, it is imperative to study the kind of transitions at a given state. We will explore the notion of control based on whether the transition is static or dynamic, or whether the values of the variable used on the transition can be tracked or not. Once we have generated \( R \) from Definition 3.2, we will perform various kinds of minimization analysis on \( R \).

**Static Transition Case:** Given a static transition, if this transition is associated with a controllable event which is allowed by the plant \( G_W \) but that violates system requirements, then we assume that this transition will be disabled by the supervisor. However, if the event associated with this transition is an uncontrollable event, then we must ensure that this static transition does not occur in the plant. For a static transition if the specification does not allow it, we will not allow the system to reach a state where it can occur. **Dynamic Transition Case:** Let \( G_{W_1} = (S_1, S_1^0, I_1, O_1, A_1, \Gamma_1, S_1^f) \) and \( G_{W_2} = (S_2, S_2^0, I_2, O_2, A_2, \Gamma_2, S_2^f) \) be two Web services, and suppose that the transition \( t_1 = s_1 \xrightarrow{m(v_1)[g_1]} s_1' \in \Gamma_1 \) is an emission of a variable \( v_1 \) from \( G_{W_1} \) to \( G_{W_2} \) and \( t_2 = s_2 \xrightarrow{m(v_1)[g_2]} s_2' \in \Gamma_2 \) is reception of \( v_1 \) by \( G_{W_2} \). Now, if the variable \( v_1 \) (the content of the message \( m(v_1) \)) has not been modified from its value at the state where it was declared until the state at which it is actually used, then the value of \( v_1 \) has not changed. This implies that we can easily track the values of the variable \( v_1 \) in the message from the service that sent it to the receiving service. Now, at the state that this variable is being used, if there is a condition on a transition \( (t_2) \) from this state that imposes a restriction on the set of values the variable can take, then we need to make sure that the guard on \( t_1 \) is never true for those values to prevent the system from reaching an illegal state. Hence, the supervisor can enact control by restricting the value of the variable at the sending service side before it would be received by the receiving service. The control strategy employed to deal with this kind of transition is to assign stronger guards to a controllable transition. Hence, we generate a guard given by \( g_1 \land \neg g_2 \) and attach it to the transition \( t_1 \). In our example above, the bank service Figure 2(c) can accept a debit card, credit card or money order as a means of payment, but the specification in Figure 3(b) prevents a payment to be made using money order. To satisfy this constraint, we will put a condition on the transition or the service that will send a request for payment to the bank service not to send a request for money order payment.

**Dynamic Nondeterministic Transition Case:** This case deals with atomic operations whose output value is unknown until runtime. This introduces an issue of nondeterminism into our model. Since the values of the variable are unknown until runtime we cannot treat this case in the same way as the previous case. During design time we will classify certain transitions (for example, failure message transitions) that depend on the output of an atomic operation as forcible events. If such a transition does not exist we will introduce a new transition into the plant and the specification. To be able to control the outputs of a given atomic operation we will rely on enforceable events to preempt uncontrollable events from happening when the output of the variable from atomic operations violates a specification. This can be done by adjustment of the plant [7], [10]. Consider Figure 2(a), the transition from state \( S_2 \) to \( S_3 \) labeled with checkAirlinesAvailable(date, loc :: \( av \)) has an output variable \( av \) which can take KLM, AirCanada or Delta as its values. The operation checkAirlinesAvailable(date, loc :: \( av \)) is assumed to be black-box, so we do not know how it assigns values to \( av \). Now, the specification in Figure 3(a) limits the values that \( av \) can take to only KLM and Delta. To ensure that the transition from state \( S_3 \) to \( S_4 \) in Figure 2(a) is never taken when the value of \( av \) is AirCanada, we mark the transition \( S_3 \xrightarrow{\text{notAvailable}} S_1 \) in Figure 2(a) as a forcible transition. The value of \( av \) is monitored so that a forcible event can be used to preempt other transitions at runtime when the value of \( av \) violates a specification.

**Definition 3.3: (Controlled System)** Let \( G_W = (S, S^0, \mathcal{I}, \mathcal{O}, \mathcal{A}, \Gamma, S^F) \) be the SLTS of a given plant, and let \( C = (S_c, S^0_c, \mathcal{I}_c, \mathcal{O}_c, \mathcal{A}_c, \Gamma_c, S^F_c) \) represent the controller for \( G_W \). The controlled system \( C \otimes G_W \) representing the behaviour of \( G_W \) when constrained (controlled) by \( C \) is given by \( C \otimes G_W = (S_c \times S, S^0_c \times S^0, \mathcal{I}_c \cup \mathcal{I}, \mathcal{O}_c \cup \mathcal{O}, \mathcal{A}_c \cup \mathcal{A}, \Gamma_c \times \Gamma, S^F_c \times S^F) \) where:

- \((s_1, s_2) \xrightarrow{m(g_1 \land g_2)} (s'_1, s'_2) \in \Gamma_c \times \Gamma_c\)
- \(s_1 \xrightarrow{m(g_1)} s'_1 \in \Gamma_c \) and \( s_2 \xrightarrow{m(g_2)} s'_2 \in \Gamma_c \)
- \(s_1 \xrightarrow{m(g_1)} s'_1 \in \Gamma_c \) and \( s_2 \xrightarrow{m(g_2)} s'_2 \in \Gamma_c \)
- \(s_1 \xrightarrow{\alpha} (s'_1, s'_2) \in \Gamma_c \times \Gamma_c \), \( \alpha \in \mathcal{A}_c \cup \mathcal{A} \) if \( s_1 \xrightarrow{\alpha} (s'_1, s'_2) \in \Gamma_c \times \Gamma_c \)
- Undefined otherwise

The first bullet states that a transition in the controlled system can be taken if an input (output) message from the controller can be consumed by an output (input) message in the plant and the guards on both the output and input transitions are satisfied. Similarly, the second bullet implies that a transition in the controlled system is possible if it is possible in both the plant and the controller transition system, which implies that the guards are true.

For a given SLTS \( T_W \) specification which is simulated by a given plant \( G_W \), a reachable state \((p^W, q^W)\) in \( G_W \times T_W \) is uncontrollable if the following holds. 1) If
an uncontrollable transition labeled with $\alpha$ (static transition) can be fired from the state $p^{GW}$ in the plant but not from the state $(p^{GW}, q^{TW})$ in $G_W \times T^W$. 2) If an uncontrollable transition labeled with $\alpha$ and a guard $g_1$ can be fired in $G_W$ at state $p^{GW}$ and this same uncontrollable transition labeled with $\alpha$ but a different guard $g_2$ is possible at $(p^{GW}, q^{TW})$ in $G_W \times T^W$, then whenever $g_1$ evaluates to true for a given set of values of a variable $v$, $g_2$ does not always evaluate to true for the same values of $v$ or simply put, the uncontrollable transition at $(p^{GW}, q^{TW})$ leads to a forbidden state.

Let $s \xrightarrow{\delta} s'$ denote that there exists at least one state $s'$ such that $s \xrightarrow{\delta} s'$ and denote $E^{P}_\delta(s) = \{a \in \Sigma \mid s \xrightarrow{a} \}$ as the set of enabled static transitions of the state $s \in S$ of the SLTS $P$. Similarly, let $s \xrightarrow{\delta[g]} s'$ and denote that there exists a guard $g$ and at least one state $s'$ such that $s \xrightarrow{\delta[g]} s'$ and let $E^{P}_\delta[s]\{a[g] \mid a \in \Sigma, g \in G\}$ represent the set of dynamic transitions enabled at state $s$ of the SLTS $P$.

**Definition 3.4:** (Controllability) Given two SLTSs $G_W = (S_W, S^0_W, T_W, O_W, A_W, \Gamma_W, S^F_W)$ and $T^W = (S^0_T \times S^1_T, O_T, A_T, \Gamma_T, S^F_T)$, representing the plant and the specification, respectively, such that $T^W \preceq G_W$. A state $(p, q) \in S^1 \times S^2$ is controllable if the following holds:
1) $\forall \delta \in \Sigma_{uc} : \delta \in E^{P}_{\delta \times T^W}(p) \implies \delta \in E^{P}_{(G_W \times T^W)}((p, q))$
2) $\forall \delta \in \Sigma_{uc} : (\delta, g_1) \in E^{P}_{\delta \times G_W}(p) \implies \exists g_2 : (\delta, g_2) \in E^{P}_{\delta \times T^W}((p, q)) \land (g_1 \implies g_2)$ or $\exists \delta', \exists g_3 : (\delta', g_3) \in E^{P}_{(G_W \times T^W)}((p, q))$ and $\delta' \in \Sigma_{fj}$

A plant $G_W$ is said to be state controllable with respect to $T^W$ if all reachable states of $G_W \times T^W$ are controllable.

**Definition 3.5:** (Non-blocking) An SLTS $G_W = (S, S^0, T, O, A, \Gamma, S^F)$ is non-blocking if $s \xrightarrow{a} s' \in \Gamma \Rightarrow \exists \delta \in \Sigma^*$ and a guard $g_j$ such that $s' \xrightarrow{\delta[g_j]} s''$ and $s'' \in S^F \setminus S^{Bad}$, where $S^{Bad}$ is the set of unsafe states. A controller is minimally restrictive in the sense that it disallows transitions only if they would violate the specification or the non-blocking condition. We formalize this qualitative property in the following definition using the preorder notion implied by simulation relation.

**Definition 3.6:** (Minimal Restrictiveness) Given a plant $G_W$ and a specification $T^W$ a controller $C$ for $G_W$ is minimally restrictive if there is no other controller $C'$ for $G_W$ such that $C \otimes G_W \leq C' \otimes G_W$.

The composition problem that we consider is as follows: given a set of available services and a set of functional requirements, construct a controller that guarantees that the requirements are satisfied. Moreover, we will seek to find a controller that is minimally restrictive and ensures that the system is free of communication errors. This is formalized in the following problem statement.

**Definition 3.7:** (Composition Problem) Let $G_{W_1}, G_{W_2}, ..., G_{W_n}$ be a set of SLTS and let $T^W$ be the specification. The composition problem is to find a non-blocking, communication-error free and minimally restrictive controller $C$ such that $C \otimes (G_{W_1} \parallel G_{W_2} \parallel ... \parallel G_{W_n}) \leq T^W$.

The definition implies that the controller constrains the plant such that every transition that can be taken by the controlled system $C \otimes (G_{W_1} \parallel G_{W_2} \parallel ... \parallel G_{W_n})$ can also be taken in the specification. The intuition is that controllability will be necessary and sufficient to solve the composition problem as formulated in the following theorem.

**Theorem 1 (Controller Existence):** Given a system modeled by an SLTS $G_W$ and a specification $T^W$ with $T^W \preceq G_W$, a controller $C$ exists such that $C \otimes G_W \preceq T^W$ if and only if $G_W$ is state controllable with respect to $T^W$. Due to space constraints, the proof is omitted and made available in [11]

**F. Composition Synthesis Algorithm**

The composition generation technique proposed in our framework is an incremental process. Algorithm 1 presents a step-by-step process that can be used to build a controllable, non-blocking and communication-error free controller. The algorithm takes the set of available component services and a goal service specifying the functional requirements as inputs. The algorithm first refines the plant $G_W$ by removing communication design errors which is given by line 2. Once the plant has been transformed into its communication-error free SLTS form, we check whether the target service is simulated by the plant $T^W \preceq G_W$ (Line 3). In the case that a simulation relation exists between the plant and the target service, we then make adjustment to the plant to include a special event that will enable the controller to enforce enforceable events at runtime. This is given by Line 4 of Algorithm 1. The algorithm then computes the composition refinement (given by Definition 3.2) of the plant and the target service to get a new SLTS $C^0$ upon which further minimization steps will be performed (line 5). The next step of the algorithm then performs various minimizations and reductions on $C^0$. Line 9 checks static controllability which is given by (first bullet of Definition 3.4). Line 10 eliminates blocking states and states that lead to only bad states. In line 11 of the algorithm we perform dynamic controllability minimization and generate stronger guards to ensure that all executions of $C^0$ lead to safe states (second bullet of Definition 3.4). Lines 7-12 perform a fixed point computation on $C^0$ and terminate when a fixed point is reached. Finally, we refine $C^k$ by removing communication errors in each trace. To ensure that the generated controller is able to communicate with the available services, we reverse the direction of the messages of $C^k$ in Line 15. In the event that the algorithm does not find a simulation relation between the plant and the target service we iteratively refine the target service until a simulation relation is found. We have applied Algorithm 1 to various small examples. Also, Algorithm 1 has been manually applied to the flight booking example introduced earlier. The asynchronous parallel product forming the plant of this example has 150 states and 3410 transitions while computing the composition refinement of the plant and the specification yielded a transition system $C^0$ with 175 states and 3945 transitions. Applying lines 6-19 of the algorithm to $C^0$ will further reduce the number of states and transitions. We hope to automate the process in the near future.
Algorithm 1 Composition Synthesizer (Controller)

Input: The SLTS representing asynchronous parallel product of the available services \( (G_{W1} \parallel G_{W2} \ldots \parallel G_{Wn} ) \) given by \( G_{W} = (S, S^0, \tau, O, A, \Gamma, S^F) \) and a target service \( T^W = (S_{T^W}, S^0_{T^W}, \Gamma_{T^W}, O_{T^W}, A_{T^W}, \Gamma_{T^W}, S^F_{T^W}) \)

Output: communication-error free, non-blocking and controllable Controller

1: procedure COMPOSER\((G_{W}, T^W)\)
2: \( G_{W} \leftarrow \text{refineCommunicationErrors}(G_{W}) \)
3: if simulationCheck\((G_{W}, T^W)\) then
4: \( G_{W} \leftarrow \text{plantAdjustment}(G_{W}) \)
5: \( C^0 \leftarrow G_{W} \times_{ref} T^W \)
6: \( k \leftarrow 0 \)
7: repeat
8: \( k \leftarrow k + 1 \)
9: \( C^k \leftarrow \text{staticControllability}(G_{W}, C^{k-1}) \)
10: \( C^k \leftarrow \text{unsafeStateMinimization}(C^k) \)
11: \( C^k \leftarrow \text{dynamicControlAndGuardGen}(C^k) \)
12: until \( C^k = \emptyset \)
13: \( C^k \leftarrow \text{removeUnsafeState}(C^k) \)
14: \( C^k \leftarrow \text{refineCommunicationErrors}(C^k) \)
15: \( C^k \leftarrow \text{reverseMessageDirection}(C^k) \)
16: else
17: \( \text{refinedPlantToSimulateTarget}(G_{W}, T^W) \)
18: Go to step 4.
19: end if
20: return \( C^k \)
21: end procedure

IV. RELATED WORK

For a more comprehensive literature review than current space allows, the reader may refer to our technical report [11]. With respect to applying DES to SOA, Wang and Nazeem [12] investigates the use of supervisory control in the artifact-centric design paradigm. They present a framework to synthesize an artifact-centric process from a given set of artifacts such that a correct execution is guaranteed by properly handling uncontrollable events. However, this work relies on the standard SCT which utilizes finite state machines for modeling. Another related work applies a variant of supervisory control theory in which system requirements are specified in modal logic to model an abstract form of service composition where non-deterministic communicating automata are used to represent Web services [13]. A lot of AI planning based approaches have been proposed to solve the problem of automatic service composition. Kazhamiakin et al. [1] present a model checking based planning approach that uses transition systems to model Web services that communicate via exchanging messages. They adopt symbolic model checking techniques into planning in order to deal effectively with non-determinism, partial observability, and complex goals. However, most of the AI planning techniques assume that the behaviour of services is deterministic and, consequently, these approaches fail when unexpected events occur. Another category of Web service composition approaches includes those that exploit transition systems and formal modeling languages such as Petri nets, UML and FSMs to model service composition [14].

V. CONCLUSION

In this paper, we have developed a supervisory control framework for modeling Web service composition. We have developed a formalism, based on SLTS, for the problem of automated Web service composition. We have developed a technique to generate a composition from a given set of Web services and a specification specified in SLTS. In this framework, we assume full observability of events but it will be of great interest to model partial observability aspects of services. Another future direction would be to determine how non-functional requirements could be incorporated into the framework. Lastly, the extension of this work to modular supervisory control will be worth investigating.

REFERENCES


