Analysis of Communication Models in Web Service Compositions *

Raman Kazhamiakin
DIT, University of Trento
via Sommarive 14
Trento, 38050, Italy
raman@dit.unitn.it

Marco Pistore
DIT, University of Trento
via Sommarive 14
Trento, 38050, Italy
pistore@dit.unitn.it

Luca Santuari
DIT, University of Trento
via Sommarive 14
Trento, 38050, Italy

ABSTRACT

In this paper we describe an approach for the verification of Web service compositions defined by sets of BPEL processes. The key aspect of such a verification is the model adopted for representing the communications among the services participating in the composition. Indeed, these communications are asynchronous and buffered in the existing execution frameworks, while most verification approaches assume a synchronous communication model for efficiency reasons. In our approach, we develop a parametric model for describing Web service compositions, which allows us to capture a hierarchy of communication models, ranging from synchronous communications to asynchronous communications with complex buffer structures. Moreover, we develop a technique to associate with a Web service composition the most adequate communication model, i.e., the simplest model that is sufficient to capture all the behaviors of the composition. This way, we can provide an accurate model of a wider class of service composition scenarios, while preserving as much as possible an efficient performance in verification.

Categories and Subject Descriptors: H.1.1 [Models and Principles]: Systems and Information Theory — *(E.4) Formal models of communication*; D.2.2 [Software Engineering]: Design Tools and Techniques; D.2.4 [Software Engineering]: Software/Program Verification — *Formal methods, Model checking*;

General Terms: Design, Verification.

Keywords: Web Service composition, BPEL, asynchronous communications, formal verification.

1. INTRODUCTION

Web services provide the basis for the development and execution of business processes that are distributed over the network and available via standard interfaces and protocols [8]. Service composition [12] is one of the most promising ideas underlying Web services: new functionalities can be defined and implemented by combining and interacting with pre-existing services. Different standards and languages have been proposed to develop Web service compositions. Business Process Execution Language for Web Services (BPEL, [2]) is one of the emerging standards for describing a key aspect for the Web service composition: the behavior of the service. It provides a core of process description concepts needed for the definition of interactions among distributed processes. This core of concepts is used both for defining the internal *business processes* of a participant to an interaction and for describing and publishing the external *business protocol* that defines the interaction behavior of a participant without revealing its internal behavior.

BPEL opens up the possibility of applying a range of formal techniques to the verification of the behavior of Web services, and different approaches have been defined for verifying BPEL [6, 13, 14, 15, 7, 17]. We are interested in particular in those techniques that are applied to the verification of BPEL compositions: in this case, we have to verify the behaviors generated by the interactions of a set of BPEL processes, each specifying the behavior of one of the services participating to the composition.

A key aspect for this kind of verification is the model adopted for representing the communications among the Web services. Indeed, the actual mechanism implemented in the existing BPEL execution engines is both very complex and implementation dependent. More precisely, BPEL processes exchange messages in an asynchronous way; incoming messages go through different layers of software, and hence through multiple queues, before they are actually consumed in the BPEL activity; and overpasses are possible among the exchanged messages.

However, most of the approaches proposed for a formal verification of BPEL compositions exploit a synchronous model of communications, which does not require message queues and hence allows for a better performance in verification. This synchronous mechanism relies on some strong hypotheses on the interactions allowed in the composition: at a given moment in time, only one of the components can emit a message, and the receiver of that message is ready to accept it (see e.g., [7]).

In our experience, these hypotheses are not satisfied by many Web service composition scenarios of practical relevance, where critical runs can happen among messages emitted by different Web services. This is the case, for instance, when a Web service can receive inputs concurrently from two different sources, or when a service which is executing

*This work is partially funded by the MIUR-FIRB project RBAU01P5SS, by the MIUR-PRIN 2004 project "Advanced Artificial Intelligence Systems for Web Services", and by the EU-IST project FP6-016004 "Software Engineering for Service-Orienrted Overlay Computers".

WWW 2006, May 23–26, 2006, Edinburgh, Scotland.
ACM 1-59593-323-9/06/0005.

a time consuming task can receive a cancellation message before the task is completed.

Our goal is to provide extended composition mechanisms, where the hypotheses on synchronous communications are weakened, but the communication model is kept as simple as possible. This way, an accurate modeling is possible for a wider class of service composition scenarios, while an efficient performance is still achievable in verification.

In this paper, we propose a model of composition, which is based on a parametric definition of the communication infrastructures. More precisely, it is possible to define different communication models by changing the number of queues existing among the component processes and the sets of messages associated with the various queues. By increasing the number of queues, and hence by allowing more and more asynchrony in the evolution of the system, we define a hierarchy of communication models that are able to model larger and larger composition scenarios. The most restrictive model, with only one shared queue of capacity 1, is shown to be equivalent to the synchronous model of [7]. The most liberal model, instead, which has dedicated queues for each type of message, can describe virtually all the examples of BPEL compositions we found in the literature and in practical usage.

The paper also describes an algorithm for the verification of BPEL compositions. The algorithm is able to identify the simplest communication model in the hierarchy that is adequate for a specific set of BPEL processes. It then builds the corresponding composition, that can be emitted in the input languages of two state of the art model checkers, namely NuSMV [4] and SPIN [9]. These model checkers can then be used to verify properties of the compositions expressed in standard specification languages such as Linear-time Temporal Logic (LTL). We conducted some experiments on our system in order to evaluate the applicability and scalability of the approach. These experiments show that the performance of the verification decreases when the complexity of the communication model increases, and that the possibility to select automatically the right model is very useful to improve the verification performance.

The paper is structured as follows. In Sect. 2 we introduce several instances of the case study that motivate the necessity to consider different variants of communication mechanism. Section 3 defines the parametric model for Web service compositions. Section 4 investigates a hierarchy of this models and addresses the problem of identifying the most adequate model for a specific composition scenario and Sect. 5 presents the algorithm for identifying and building such most adequate model. An experimental evaluation of the approach is presented in Sect. 6, while conclusions and future work are presented in Sect. 7.

2. BPEL COMPOSITION SCENARIOS

In order to illustrate the problem of modeling service compositions, we consider several variants of the Virtual Travel Agency domain. The goal of the Virtual Travel Agency is to provide a combined flight and hotel booking service by integrating two independent existing services: a Flight booking service, and a Hotel booking service. Thus, the composition describes the interactions of four partners: User, Virtual Travel Agency (VTA), Hotel and Flight services (see Fig. 1.a).

In our framework, we model the composition using BPEL

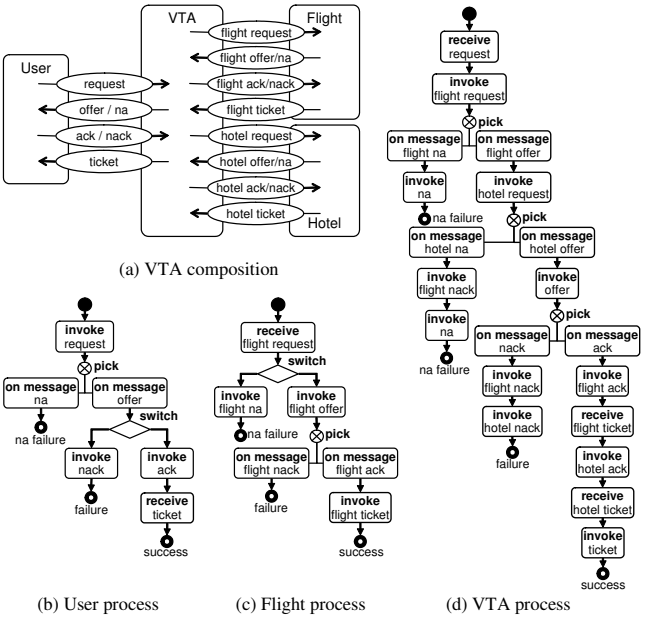

Figure 1: Composition participants

specifications that describe the workflows and the interactions of the four partners. BPEL provides an operational description of the (stateful) behavior of web services on top of the service interfaces defined in their WSDL specifications. An abstract BPEL description identifies the partners of a service, its internal variables, and the operations that are triggered upon the invocation of the service by some of the partners. Operations include assigning variables, invoking other services and receiving responses, forking parallel threads of execution, and non-deterministically picking one amongst different courses of actions. Standard imperative constructs such as if-then-else, case choices, and loops, are also supported.

In this paper, we will use diagrams like the ones in Fig. 1 for representing the BPEL specifications. The BPEL sources for the examples we describe in the paper are available at `http://www.astroproject.org/`.

2.1 Example 1: Tickets Reservation Scenario

In this scenario, the user can ask the VTA to book a flight to a specified location and reserve a room in a hotel at that location for a given period of time. It is possible that the request of the user cannot be fulfilled, in which case the user receives a not-available (`na`) notification from the VTA. If a reservation offer is received instead, the user can accept or reject it, sending a corresponding message to the VTA (Fig. 1.b).

The Flight booking service becomes active upon a request for a given location (e.g., Paris) and a given period of time (e.g., August). In the case the booking is not possible, this is signaled to the requestor, and the protocol terminates. Otherwise, the requestor is notified with an offer information and the protocol stops waiting for either a positive or negative acknowledgment. In case of positive answer, the flight is successfully booked and the reservation ticket is sent, otherwise the interaction terminates with failure. Figure 1.c represents the protocol provided by the Flight booking service. The protocol of the Hotel service is similar.

The behavior of the VTA is as follows. After receiving a reservation request from the user, the VTA interacts with Flight and Hotel services to obtain ticket offers and expects either a negative answer if this is not possible (in which case the user is notified and the protocol terminates with failure), or provides the user with an offer indicating hotel, flights and cost of the trip. After that, the user may either accept or refuse the offer, and in the first case VTA provides the user with the tickets obtained from Hotel and Flight. The diagram corresponding to the BPEL protocol of VTA is represented in Fig. 1.d.

This composition scenario exhibits an important property that allows for a very simple communication mechanism. At any moment of time, only one of the partners is ready to emit a message. Moreover, the corresponding receiver is ready to accept the message. Using the terminology of [7], the composition model satisfies the *synchronous compatibility*, *autonomy* and *lossless composition* properties. As a consequence, a synchronous communication model can be used to define the composition without loosing completeness of behaviors. As demonstrated in [7], this allows for an efficient verification of the composition scenarios.

2.2 Example 2: Reservation with Cancellation

Unfortunately, the simplified communication model of the previous example is not applicable to all kinds of interactions. A typical example is a business process with event handlers. Let us consider an extension of the ticket reservation scenario, such that the user can decide to cancel the booking operation. In this case the user can send a `cancel` message to the VTA and wait for the outcome of the cancellation. The VTA forwards the cancellation to the Flight (and similarly to the Hotel process, we omit this for the sake of simplicity). The Flight waits for a cancellation message for a certain time after the acknowledgement of the reservation. If the cancellation message is received on time, the Flight notifies a successful cancellation. If the time for a cancellation runs out, the Flight sends a ticket to the VTA, thus forcing the failure of cancellation: cancellations sent by the VTA after the ticket is sent are consumed and ignored. The excerpts of the corresponding process specifications are represented in Fig. 2.a.

The verification under the synchronous communication model is not able to manage correctly this example, and reports a deadlock. Indeed, if the Flight service fails to wait for a cancellation, the `onAlarm` activity is fired and a ticket is sent to the VTA process. Meanwhile, the VTA may receive a cancellation message from user and forward it to the Flight service. Therefore both Flight and VTA would try to send messages to each other and the composition would be in a deadlock according to the synchronous semantics.

This deadlock is not real, in the sense it does not occur in existing BPEL engines; since the Web services communications are asynchronous, and the message emission is not blocking, both processes will emit messages to each other. Both messages will be consumed then and the composition terminates correctly.

The problem we are facing here is that the synchronous model is too restrictive. The message delivery and processing may require a certain time, thus leading to situations where concurrent message emissions take place. These situations, however, are not allowed in the synchronous communication model. In order to verify correctly the considered

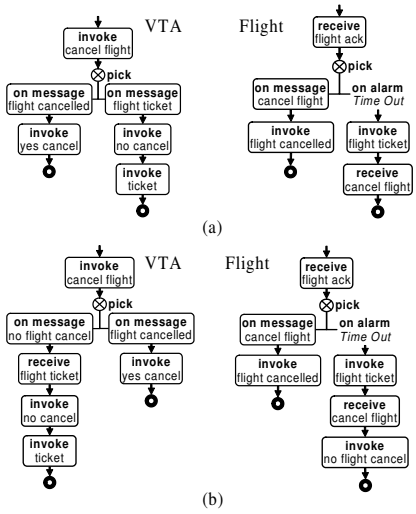

Figure 2: Cancellation management

example, a relaxed model is needed that allows considering these concurrent message emissions.

If one applies the verification approach of [7] for the analysis of such a composition, then the inapplicability of the synchronous communication model is detected and reported. However, [7] and other verification approaches fail to find an alternative communication model that is adequate to the scenario. In [7], an arbitrary communication model with 2-position buffers is applied, which is rather expensive from the verification point of view, and still does not guarantee the correctness of the verification.

2.3 Example 3: Extended Cancellation

Let us consider a further modification of the case study. In this case, after the VTA has sent a cancellation message to the Flight, it waits for a message notifying whether the cancellation is possible (message `flight cancelled`) or not (message `no flight cancel`). In the latter case, it waits for the ticket and sends is to the user. The Flight service, on the other side, behaves as before with the only difference that, after emitting the ticket and receiving the cancellation, it sends a notification about cancellation rejection (message `no flight cancel`). The corresponding diagrams are represented in Fig. 2.b.

Even if one verifies the example allowing for concurrent message emissions, the following incorrect scenario may occur. The Flight service sends a ticket and waits for a cancellation. At the same time, the VTA process sends a cancellation request that the Flight service rejects. The VTA has received a ticket and then a cancellation rejection, but it is not able to process the messages in this order. Only if the execution of processes in the run-time environment allows for reordering of messages (which is the case for existing implementations) the deadlock disappears, since the cancellation rejection can be processed before the ticket message.

This example shows a necessity not only to consider systems which do not follow the synchronous communication semantics, but also to accept less restrictive models where message reordering is allowed. If this is not done, then scenarios that can occur in practice are not considered in the verification, and wrong results can be obtained.

2.4 Assumptions on the Composition

This chain can be further prolonged, leading to more complex communication models. One can think of lossy channels, complex ordering conditions, complex queue models etc. As a result, the behavior of the given composition varies when different models are applied. In the following we present a parametric communication model that is suitable for the description of wide class of composition scenarios, including those discussed in Examples 1, 2, 3. This model, however, relies on certain hypotheses on the Web service compositions. These hypotheses define some boundaries to the scope of our verification framework, and will allow us to abstract from low-level issues that are irrelevant for the "logic" of the composition, and to simplify the formalization of the model:

- The communication channels used by the participants are disjoint. That is, it is never the case that two processes are able to invoke or receive the same operation from a third process. Formally, this means that the sets of communication actions used by pairs of processes are disjoint, and that the end points of the communications are statically fixed.
- The communication channels are perfect, i.e. no message losses can ever occur. This assumption may be enforced by special techniques in the WS world. In particular, WS Reliable Messaging or monitoring techniques can be used for these purposes.
- The definition of the composition participant should not include an infinite loop of internal actions. Indeed, such a loop would describe a divergent behavior of the system, i.e., a behavior where the service is not interacting with the environment.

Under these assumptions, we consider a very general communication infrastructure that allows for modeling the following features:

- Invoke operations are non-blocking. Due to the asynchronous, loosely coupled nature of Web services, the message emission cannot be blocked even if the receiver is not ready to accept the message.
- Queue are unbounded, but with the restriction that the number of messages in the queues cannot grow unboundedly. That is, we do not define a limit to length of the queues "a priori", however we consider invalid all those composite systems where the number of messages in the queue can grow unboundedly. (This corresponds to assume that the queues are "long enough" to contain all the messages that need to be stored in the executions of the systems.)
- An arbitrary implementation of the underlying queuing mechanism is allowed. That is, we do not commit to a specific model of implementation of the queues. In order to reflect this property, we assume the most general behavior of queue, where the messages can be consumed in any arbitrary order, regardless the order in which they are stored in the queues.

We remark that the most critical requirement is the last one. Indeed, existing BPEL engines manage queues in a specific way, and some of the behaviors that are possible with the arbitrary queues could not be possible in the concrete implementation of queues of a specific engine. Assuming the most general behavior of queues allows us to guarantee that the theoretical model includes all behaviors that are possible in every specific engine. We will see that, for certain classes of systems, it is possible to identify some properties of the engine that guarantee that all behaviors allowed by the theoretical model can occur in the concrete implementation.

3. WS COMPOSITION FORMALIZATION

We now present the formal model we propose for representing and analyzing Web service compositions. We provide a formal representation of stand-alone Web services and generic Web service compositions. We also define the behavior of the composition used as a basis for the composition verification.

3.1 BPEL as State Transition System

In our framework, we encode BPEL processes as *state transition systems* which describe dynamic systems that can be in one of their possible *states* (some of which are marked as *initial states*) and can evolve to new states as a result of performing some *actions*. We model process interactions as *external actions* defined on a set of operations (or message types) M. Following the standard approach in process algebras, external actions are distinguished in *input actions*, which represent the reception of message of type $\alpha \in M$, denoted as $\overleftarrow{\alpha}$, and *output actions*, which represent messages of type $\alpha \in M$ sent to external services, denoted as $\overrightarrow{\alpha}$. We also define a special action τ, called *internal action*, which is used to represent evolutions of the system that do note involve interactions with the external services. A *transition relation* describes how the state can evolve on the basis of inputs, outputs, or of the internal action τ.

Definition 1. A *state transition system* (STS) is a tuple $\langle \mathcal{S}, \mathcal{S}_0, \mathcal{I}, \mathcal{O}, \mathcal{R} \rangle$ where $\mathcal{S}$ is the finite set of states and $\mathcal{S}_0 \subseteq \mathcal{S}$ is the set of initial states; $\mathcal{I}$ is a finite set of input actions and $\mathcal{O}$ is a finite set of output actions; $\mathcal{R} \subseteq \mathcal{S} \times (\mathcal{I} \cup \mathcal{O} \cup \{\tau\}) \times \mathcal{S}$ is the transition relation.

The assumption on the finiteness of the states set is required in order to enable the analysis techniques presented in this work. For the sake of space, we omit the discussion on the formal translation from BPEL to STS. This translation is implemented inside our toolkit, which is available from `http://www.astroproject.org/`.

3.2 State Transition System with Channels

In order to represent a composition of Web services, we now define *state transition systems with channels*. This model describes the executions of the composition according to a parametric definition of the communication model.

Intuitively, this model is characterized by a set of global states, describing the composition during its execution, and a set of (FIFO) queues that store the messages exchanges among partners[1]. A global state contains two components: a *control state* that represents the local states of the participating STSs, and a *queue content* that defines sets of

[1]We remark that in spite of the fact that the queues are ordered, there is still a possibility to define an unordered message consumption. In the following sections we will show how this can be done in our framework.

messages stored in the queues in a particular moment of time.

More precisely, let us assume that the composition is built from n STSs $\langle \mathcal{S}^i, \mathcal{S}^i_0, \mathcal{I}^i, \mathcal{O}^i, \mathcal{R}^i \rangle$ representing the participating Web services. We represent a control state as a vector $S = \langle s_1, \ldots, s_n \rangle$, where s_i is a local state of the i^{th} STS. We denote a vector with component s_i updated to s'_i as $S[s_i/s'_i]$. Let us also assume to model the communications among Web services with set of m queues with disjoint alphabets $M_j \subseteq M$, $1 \leq j \leq m$. A queue q_j may be declared as *bounded*, with the corresponding capacity $0 < b_j < \infty$, or *unbounded*, in which case $b_j = \infty$. As one can see, given the same set of STSs, different configurations may be used to represent their composition. These configurations are parametric with respect to the number of queues, to the distribution of the queue alphabets, and to the queue bounds. We denote such configurations as *communication models*.

Definition 2. A *communication model* for the STS composition is a tuple $\Delta = \langle B, \mathcal{L}_M \rangle$, where $B = \langle b_1, \ldots, b_m \rangle$, is a vector of queue bounds, and $\mathcal{L}_M : M \rightarrow [1 \ldots m]$ is a function that associates a message type α with a queue $i = \mathcal{L}_M(\alpha)$. The alphabet M_i of queue i is defined as follows: $M_i = \{\alpha \mid \mathcal{L}_M(\alpha) = i\}$.

We define a queue content as a vector $C = \langle w_1, \ldots, w_m \rangle$, where $w_j \in M_j^*$ represents a sequence of messages stored in the j^{th} queue. An operator . denotes the concatenation. That is, $C.\alpha = \langle w'_1, \ldots, w'_m \rangle$, where $w'_j = w_j.\alpha$ if $\alpha \in M_j$, and $w'_j = w_j$ otherwise. We write $|C| \leq B$ to specify that $|q_i| \leq b_i$.

Definition 3. A *State Transition System with Channels* (CSTS) for the composition of n STSs under a communication model $\Delta = \langle B, \mathcal{L}_M \rangle$ is a transition system $\Sigma_\Delta = \langle GS, gs_0, A, T \rangle$ where

- GS is a set of *global states* of the form $\langle S, C \rangle$, where $S = \langle s_1, \ldots, s_n \rangle$, with $s_i \in \mathcal{S}^i$, is a control state, and $C = \langle w_1, \ldots, w_n \rangle$, with $w_i \in M_i^*$ and $|w_i| \leq b_i$, represents the contents of the queues;
- $gs_0 = \langle S_0, \langle \epsilon, \ldots, \epsilon \rangle \rangle$ is an initial global state;
- A is a set of actions $a \in \{\cup_i \mathcal{I}^i \bigcup \cup_i \mathcal{O}^i \bigcup \{\tau\}\}$.
- $T \subseteq GS \times A \times GS$ is the global transition relation. A transition $t = (gs, a, gs')$ is in T, if for some $1 \leq i \leq n$, $S' = S[s_i/s'_i]$ and $(s_i, a, s'_i) \in \mathcal{R}^i$, and one of the following holds:
  - $a = \overrightarrow{\alpha} \ \wedge \ C(gs') = C(gs).\alpha \ \wedge \ |C(gs')| \leq B$;
  - $a = \overleftarrow{\alpha} \ \wedge \ C(gs) = \alpha.C(gs')$;
  - $a = \tau \ \wedge \ C(gs') = C(gs)$.

The behavior of CSTS can be described with a directed (possibly infinite) labeled tree, called *reachability tree RT*. Nodes in this tree are labeled with (reachable) global states $gs \in GS$; the root is labeled with the initial global state gs_0; edges are labeled with actions $a \in A$. The *reachability graph RG* is obtained from *RT* by merging nodes labeled with identical global states.

We say that the CSTS is *complete* if all the terminating global states $\langle S, C \rangle$ (that is, leaves of the reachability tree) have empty queue content: $C = \langle \epsilon, \ldots, \epsilon \rangle$. We remark that systems that are not complete loose message: indeed, at the end of the computation there are unconsumed messages in queues. We will consider only complete CSTS in the following.

We say that action $a \in A$ is fireable in state gs, if there is a transition $(gs, a, gs') \in T$. In this case, we write $gs \xrightarrow{a} gs'$. Let $\omega = gs_1, a_1, gs_2, a_2, \ldots$ be a (possibly infinite) sequence of states and actions interleaved. We say that the sequence is fireable from gs_1, written as $gs_1 \overset{\omega}{\rightarrow}^*$, if $\forall k \geq 1, \ gs_k \xrightarrow{a_k} gs_{k+1}$. The *behavior* of the CSTS is a set of such sequences fireable from the initial state:

$$\Omega = \{\omega \mid gs_0 \overset{\omega}{\rightarrow}^*\}$$

The behavior of the CSTS describes possible executions of the composition and is used as a formal basis for various verification techniques. In particular, we allow for reachability analysis, and for the verification of temporal specifications expressed as Linear-time Temporal Logic (LTL) formulas [5] evaluated on the CSTS behavior.

Since we assume finite-state BPEL processes, the control states of a CSTS are also finite. The only possibility to have an infinite reachability graph is when the messages contained in a channel can grow unboundedly. We have already discussed in Sect. 2.4 that we consider systems with channels that grow unboundedly as "bad" systems. Moreover, finite-state verification is not applicable on these systems. Therefore, one of the critical problems to be addressed during the analysis is to identify and rule out these systems. In the following, we say that the channel of a CSTS have a *bounded growth* if, for every queue q_i, either a finite bound $b_i < \infty$ is declared, or there is some constant K_i such that the queue contains at most K_i messages in all reachable states.

We remark that the proposed CSTS-based model of Web service composition captures only the control flow of the participating processes. However, BPEL allows one to define also the data flow of the composition. In other words, it is possible to define the data values carried by the messages, conditions on the process transitions etc. This information has to be taken into account in the analysis of the BPEL composition. As we show later, this is addressed by enriching the composition with variables and other data-dependent constructs before passing it to the model checker, and performing the actual verification on the enriched models. The presence of data values in messages also influences the alphabet on which the queues are defined. Indeed, the management of queues could depend not only on the type of the messages, but also on the associated values. This may be resolved refining the model by introducing a new message type for each distinct set of data values or, for each set of data values that can be distinguished by the BPEL run time environment. If the data domains are finite, the approach we describe below works also in this refined setting. We intend to better investigate in future works the issues related to an adequate model of data domains in the definition of the queue structure.

4. HIERARCHY OF MODELS

The definition of CSTS is parametric w.r.t. a communication model. Different communication models (and hence queue structures) define different behaviors for the same composition scenario. Therefore, the result of the verification of a composite system depends on the selected com-

munication model. In order to guarantee the correctness of the verification, we have to make it sure that the selected communication model allows for all the behaviors that are compatible with the assumptions presented in Sect. 2.4.

In this section we address the problem of defining suitable communication models for CSTS, and of guaranteeing that these communication models are *adequate* w.r.t. the real executions, i.e., that they do not discard any execution that can happen according to our assumptions.

This is achieved through the following steps:

- We define the "most general" model in terms of CSTS. This is a model which allows more behaviors than any other communication model does.
- We define a family of possible communication models that we can adopt for the verification of composite systems. These communication models correspond to different levels of complexity and efficiency of the verification process. All the models we propose are expressible in terms of CSTS, by changing the queue model.
- We define the "adequacy" of a communication model for a composite system: a communication model is adequate if it expresses all the behaviors of the most general model, i.e., no behaviors are lost due to the specific queuing model.
- For each communication model, we also discuss the requirements on the middleware that guarantee that all the behaviors expressed by the model can happen in real implementations of BPEL engines.

4.1 Relations among Communication Models

One of the tasks in the adequacy analysis is to check whether the composition of Web services under the given communication model does not loose behaviors w.r.t. some other more general model. This requires introduction of certain relations between models, namely *simulation* relations.

We will write Σ_Δ to denote the composition of a given set of STSs under communication model Δ.

Definition 4. We say that Σ_{Δ_2} *simulates* Σ_{Δ_1}, written as $\Sigma_{\Delta_1} \preceq \Sigma_{\Delta_2}$, if $\Omega_{\Sigma_{\Delta_1}} \subseteq \Omega_{\Sigma_{\Delta_2}}$.

We say that Σ_{Δ_1} and Σ_{Δ_2} are *bisimilar*, written as $\Sigma_{\Delta_1} \approx \Sigma_{\Delta_2}$, if $\Omega_{\Sigma_{\Delta_1}} = \Omega_{\Sigma_{\Delta_2}}$.

When the simulation relation among two communication models Δ_1 and Δ_2 holds for any set of STSs, we say that the model Δ_2 is *more general* than the model Δ_1.

Definition 5. Communication model Δ_2 *simulates* model Δ_1, written as $\Delta_1 \sqsubseteq \Delta_2$, if for any composition of STSs, $\Sigma_{\Delta_1} \preceq \Sigma_{\Delta_2}$.

Being reflexive and transitive, this relation forms a partial order on the set of communication models. Below we will show that there is a "most general" model, that is the model $\overline{\Delta}$, such that for any other model Δ holds $\Delta \sqsubseteq \overline{\Delta}$.

The relation among communication models relies on the structure of the queues. There are two dimensions in which the models differ. First, the relation depends on the queue bounds: the bigger a queue bound is, the more transitions might be enabled. Second, it depends on the distribution of the message alphabets: if the alphabet of each queue in one model is a subset of the alphabet of some queue in another model, then the first model is more general then the other. The following theorem defines relation between models with different queue structures.

THEOREM 1. *Consider two communication models* $\Delta_1 = \langle B_1, \mathcal{L}_{1M} \rangle$ *and* $\Delta_2 = \langle B_2, \mathcal{L}_{2M} \rangle$. *If for each queue* q_{2i} *there exists a queue* q_{1j}, *such that*

- $\forall\ \alpha \in M.\ \mathcal{L}_{2M}(\alpha) = i\ \Rightarrow\ \mathcal{L}_{1M}(\alpha) = j$, *and*
- $b_{1j} \leq b_{2i}$

then $\Delta_1 \sqsubseteq \Delta_2$.

The theorem may be easily understood on the following example. Consider a model Δ_1 with one queue q_1 with alphabet $M = \{\alpha_1, \alpha_2\}$, and a model Δ_2 with two queues q_{21} and q_{22} with alphabets $\{\alpha_1\}$ and $\{\alpha_2\}$ respectively. Indeed, if an input action is allowed in the composition under model Δ_1 then it is also allowed in the second model, since if a message is on the top of the queue in first model and can hence be consumed, then it is on the top of one of the queues in the second model. Similarly, if an output action $\overrightarrow{\alpha_1}$ is allowed in the first model, then the queue is not full, that is $|q_1| < b_1$. Since $|q_{21}| \leq |q_1|$ (the two queues have the same length if q_1 contains only messages of type α_1) and, by hypothesis $b_1 \leq b_{21}$, then $|q_{21}| < b_{21}$ and hence the output action $\overrightarrow{\alpha}$ is not blocked in the second model.

4.2 Most General Communication Model

The first step of the adequacy analysis is to define the reference model, that is, the model that allows for the largest set of behaviors. In order to respect the assumptions of Sect. 2.4, this model has to allow for potentially unbounded queues, non-blocking emissions, and arbitrary, unordered access to the content of any queue.

The definition of this model is based on the following observation. Two invocations of the same operation lead to a situation where the queue contains two messages of the same type. From the external point of view, the two messages are indistinguishable. The same holds for the possibility to consume them in any order: if two messages of the same type are contained in the queue, then the order in which they are consumed becomes irrelevant. Therefore, in order to model arbitrary access to any message in any queue, it is enough to model such a system with the separate queue for each message type.

Definition 6. The *Most General Communication Model* (MG-model) for the composition of n STSs is a model $\overline{\Delta} = \langle B, \mathcal{L}_M \rangle$, with $|M|$ queues, $b_i = \infty$, and $\mathcal{L}_M(\alpha_i) = q_i$.

It is easy to see that such a model is indeed a generalization of any other communication model w.r.t. the behavior of any composition of STSs.

Proposition 1. For any communication model Δ, $\Delta \sqsubseteq \overline{\Delta}$, where $\overline{\Delta}$ is MG-model.

Whenever a composition under a certain model Δ simulates the most general composition, we say that this model is *adequate* for the description of the composition scenario.

Definition 7. A communication model Δ is said to be *adequate* for the given composition scenario if $\Sigma_\Delta \approx \Sigma_{\overline{\Delta}}$.

An important result in the proposed framework is that the finiteness of the reachability graph under the MG-model is decidable. Indeed, the class of systems that can be modeled in this way forms a subclass of Petri-Nets, and "boundedness" is decidable for Petri-Nets [10].

Proposition 2. The finiteness of the reachability graph of the composition of STSs under the MG-model is decidable.

Model $\overline{\Delta}$ defines the most liberal policy for the message processing: each message stored can be accessed and consumed regardless the reception order. On the other hand, this model is also the least realistic, among the ones described in this section, for what concerns the implementation of a middleware generating all the behaviors allowed by the model. Indeed, all existing engines apply a specific policy for the queues and do not allow for such an arbitrary consumption of messages as the one allowed in the model.

4.3 Communication Models Interpretation

We now define a hierarchy of communication models that are particularly significant for verifying Web service compositions and that have been proposed in the literature.

4.3.1 Synchronizable Communications.

This is the most restricted communication model that can be defined in terms of CSTS formalization. In this model there is only one queue of capacity one.

Definition 8. The *synchronizable communication model* is $\Delta_1^1 = \langle B, \mathcal{L}_M \rangle$, with $B = \langle 1 \rangle$ and $\mathcal{L}_M(\alpha) = 1$ for all messages α.

This model is strongly related to another communication model widely used for modeling Web service compositions, namely *synchronous composition*. In such a model, communicating processes synchronize on shared actions; therefore this model can be represented without queues. More precisely, when the Δ_1^1 model is shown to be adequate for a given composition scenario, one can use a synchronous composition for the analysis of wide range of properties, thus achieving better performance. In the following, we formalize this result. We start with a definition of a synchronous composition.

Definition 9. A *synchronous composition* Σ_s of n STSs is a tuple $\langle GS, gs_0, A, \mathcal{R} \rangle$ where

- GS is a set of global states $gs = \langle s_1, \ldots, s_n \rangle$; gs_0 is the vector of initial states of the STSs;
- $A = \{\tau\} \cup \{\alpha \mid \exists\ i\ .\overrightarrow{\alpha} \in \mathcal{O}^i\}$ is a set of actions;
- A transition $t = (gs, a, gs')$ is in $\mathcal{R}$, iff
  - $a = \tau$, for some $1 \leq i \leq n$, $gs' = gs[s_i/s_i']$, $(s_i, \tau, s_i') \in \mathcal{R}^i$
  - $a = \alpha$, $gs' = gs[s_i/s_i', s_j/s_j']$, $(s_i, \overrightarrow{\alpha}, s_i') \in \mathcal{R}^i$, and $(s_j, \overleftarrow{\alpha}, s_j') \in \mathcal{R}^j$.

Let us define a *conversation* of the composition as a sequence of messages emitted during interactions [7]. That is, $\gamma = \alpha_1, \alpha_2, \ldots$, with $\alpha \in M$, is a conversation if there is a behavior $w = \omega = gs_1, a_1, gs_2, a_2, \ldots$, and for each $i > 0$ there exists $j > 0$ s.t. $gs_j \xrightarrow{a_j} gs_{j+1}$ and $a_j = \overrightarrow{\alpha_i}$. We denote a conversation set of the composition as Γ.

When the verification properties are defined on the set of conversations, and the composition appears to be complete under Δ_1^1 model, then one can use the synchronous product for the composition analysis.

THEOREM 2. *Let Σ_1^1 and Σ_s be a complete composition of n STSs under Δ_1^1 model and their synchronous product respectively, with the corresponding conversation sets Γ_1^1 and Γ_s. Then $\Gamma_1^1 = \Gamma_s$.*

Due to the strong hypotheses on the synchronizable communication model, the kinds of systems for which the model is adequate are also subject to restrictive hypotheses on the kinds of interactions that can occur. As a consequence, the compositions, for which this model was proved to be adequate, are very robust and exhibit the same behavior on all the implementations of BPEL engines. For this reason, the synchronizable model is the less demanding on the underlying middleware among the ones studied in this paper.

4.3.2 Locally Ordered Asynchronous Communications.

This model is used in some works for the representation of WS compositions (see e.g. [7]). Each participant is equipped with separate queue storing messages from all the partners.

Definition 10. A *locally ordered asynchronous communication model* of n STSs is $\Delta_{lo} = \langle B, \mathcal{L}_M \rangle$, with n queues, $b_i = \infty$, and $\forall\ \alpha$, s.t. $\overleftarrow{\alpha} \in \mathcal{I}^i$. $\mathcal{L}_M(\alpha) = q_i$.

This model is more general than the synchronizable model:

$$\Delta_1^1 \sqsubseteq \Delta_{lo}\ .$$

Indeed, this model is required for describing the composition scenario in Sect. 2.2.

This communication model requires that messages are queued on a process-by-process way. This policy for managing queues is a reasonable and easy to implement, and it provides a good compromise between the complexity of the implementation and the class of examples it is able to cover. Similar considerations also hold for the next model we present.

4.3.3 Mutually Ordered Asynchronous Communications.

In this model, a pair of queues is defined for each pair of processes, with each queue representing one direction of interaction between these processes. This model, described in [3], provides a natural representation of communicating BPEL processes since each process explicitly distinguishes each of its partners. The main feature of this model is that each pair of communicating processes preserves the order of partners' events. In other words, the order of receptions is equivalent for each pair of processes.

Definition 11. A *mutually ordered asynchronous communication model* of n STSs is $\Delta_{mo} = \langle B, \mathcal{L}_M \rangle$, with $n^2 - n$ queues denoted as $q_{i,j}$ $(i \neq j)$, s.t. $b_{i,j} = \infty$, and $\forall\ \alpha$. $\overleftarrow{\alpha} \in \mathcal{I}^j \wedge \overrightarrow{\alpha} \in \mathcal{O}^j$ iff $\mathcal{L}_M(\alpha) = q_{i,j}$.

This model is clearly more general than the synchronous model. It is also more general with respect to locally ordered asynchronous model. To see this, notice that the input messages of the particular process are stored potentially

in several queues instead of only one. Therefore, the alphabet of these queue is smaller, and more actions are fireable (Theorem 1). This model is required for the composition scenario described in Sect. 2.3.

We conclude this section with the overall hierarchy of the models defined:

$$\Delta_1^1 \sqsubseteq \Delta_{lo} \sqsubseteq \Delta_{mo} \sqsubseteq \overline{\Delta} \ .$$

We remark that the CSTS formalism allows for potentially infinite number of models to be defined. The MG-model is the upper bound of this construction and we use this fact in the adequacy analysis presented below.

5. BUILDING AN ADEQUATE MODEL

We now present an approach for the analysis of compositions of Web services. In this approach, we incrementally pass through the models starting from the synchronizable until the least general adequate model is found for the given composition scenario. As we will see in the Sect. 6, this allows not only to find a proper model of communication for the scenario but also to perform the analysis more efficiently. Indeed, if the model is shown to be adequate for the given composition, and the composition behaves correctly, then it will be correct also under more general models.

The number of models that we could consider in our approach is potentially infinite. Here, we assume to have fixed a finite set of models that we consider interesting for the analysis (this could be for instance the sequence of models we have introduced in the previous section). We assume moreover that the simulation relation defines a total order on these models, and that the MG-model belongs to the set (and is hence its upper bound). More precisely, the algorithm of the approach is as follows:

1. take a sequence of models $\Delta_1, \Delta_2, \ldots, \overline{\Delta}$ such that $\Delta_i \sqsubseteq \Delta_{i+1}$;
2. analyze the models until the adequate one is found: $\Sigma_{\Delta_i} \approx \Sigma_{\overline{\Delta}}$;
3. the composition is checked for completeness (i.e., the queues are empty in the terminal states of the composition) and bounded growth.

When an adequate communication model is identified, and the composition is shown to have queues with a bounded growth, the obtained reachability graph may be used as a basis for further verification tasks. Indeed, the graph is finite, and actual queue bounds may be extracted by analyzing reachable states thus allowing for finite representation of the composition model in the model checker specifications.

As we mentioned above, the results of the verification may be affected by the data flow specified in the BPEL code. For this reason the composition obtained after steps specified above is equipped with data-related constructs, and the resulting model is analyzed using model checking techniques.

5.1 Algorithm

The algorithm is used to give an answer for the following questions: *(i)* the model under consideration is adequate for the description of the given composition; *(ii)* the composition is complete and has queues with a bounded growth.

The algorithm is presented in Alg. 1. The outcome of the algorithm is the constructed reachability graph representing

Algorithm 1 Composition adequacy check

```
 1: States := nil;      {Stack of states}
 2: Visited := nil;     {Set of visited states}
 3: Transitions := nil; {Set of transitions}
 4: IS := nil;          {Set of incomplete states}
 5: US := nil;          {Set of unbounded states}
 6: explore(gs_0);
 7: procedure explore(s)
 8: push(s, States);
 9: current := s;
10: Fireable := fireable(current); {fireable actions}
11: if Fireable ≠ fireable_MG(current) then terminate;
12: Transitions := Transitions ∪ Fireable;
13: if Fireable ≠ ∅ then
14:     forall transition ∈ Fireable do
15:         s' := transition.target;
16:         if s' ∉ States ∪ Visited then
17:             if ∃ ss ∈ States s.t. U(ss, s') then
18:                 US := US ∪ {current};
19:             else explore(s');
20: else if ¬emptyChannels(current) then
21:     IS := IS ∪ {current};
22: Visited := Visited ∪ {current};
23: pop(s, States);
24: end procedure
```

the composition, if the model is adequate, or the witness of the fact that the model is not adequate. Whenever the state with non-empty queue content is found s.t. it can be never completed, it is added to the special container *incomplete*.

In order to terminate the search when the composition has unbounded growth, we use the following relation U:

Definition 12. If s and s' are two nodes of the reachability tree labeled with global states gs and gs' and such that $s \xrightarrow{w}{}^{*} s'$, where w is a sequence of actions.

$$U(s, s') \Leftrightarrow S(gs) = S(gs') \ \wedge \ C(gs) \preceq C(gs')$$

Here we write $C(gs) \preceq C(gs')$ if for any queue index i there is a suffix $\bar{w}_i$ such that $w'_i = w_i.\bar{w}_i$. Thus two states are in the relation iff the control state is the same and the queue content increased. We write $U_{\neq}(s, s')$ to denote that $U(s, s')$ and $C(gs) \neq C(gs')$.

THEOREM 3. *Let s and s' are two nodes of the reachability tree for the composition under the MG-model $RT(\Sigma_{\overline{\Delta}})$. If $s \xrightarrow{w}{}^{*} s'$ and $U_{\neq}(s, s')$ then the reachability graph $RG(\Sigma_{\overline{\Delta}})$ is infinite.*

The behavior of the algorithm is the following.

- A set of containers to store the reachability graph structure and the "bad" states (i.e. unbounded or incomplete) is defined (lines 1-5).
- The recursive procedure *explore* that implements a DFS-algorithm is defined (lines 7-24) and called for initial state (line 6).
- The current state is added to the stack and a set of fireable actions from this set is extracted (line 10). If this set is not equivalent to the set of actions fireable under MG-model ($firable_{MG}$), then the algorithm terminates since the selected model is not adequate (11).

- If the set of actions is not empty, then we analyze the target state s' of each action (14-19).
- If the state is fresh then we check the unboundedness by looking for a cycle that increments queue content (17-18). If there is no such a cycle, we call *explore* procedure recursively.
- If the set of fireable actions is empty then we reached the leaf of the search tree and just check that all the messages are consumed (20-21).
- The current state is thus explored; we add it to the set of visited states, and remove from the stack.

5.2 Adding Data to the Composition Model

When the reachability graph that represents the behavior of the composition is constructed, we enrich it with the data-related part from the component processes. BPEL allows for the description of the data flow in the processes by defining variables of arbitrary types and operations on them. In particular, one can assign a certain value to the variable, use it as a parameter for the remote operation invocation, or use its value in the conditions in *if-then-else* constructs or loops. As a result, in the model with data the executability of certain transitions depends on the values of these variables, and the behavior of the composition may change affecting the analysis results.

For this reason actual verification is performed on the models equipped with data using model checking techniques. In order to be able to apply them, such an extended model has to be finite, and therefore the domains of the variables should also be finite. Detailed description of the data-related constructs translation and manipulation may be found at `http://www.astroproject.org/`.

6. EXPERIMENTAL RESULTS

A prototype of a verification tool based on the parametric communication model presented in this paper has been implemented within the Astro toolkit and is available as part of the project (`http://www.astroproject.org`).

We conducted series of experiments in order to evaluate the presented approach. The aim of the evaluation was to demonstrate that the less general model shown to be adequate is more efficient for the analysis, and to see the overall performance of the composition verification based on the presented approach.

In particular, we were interested in the performance and in the memory usage of the composition analysis. In these experiments we used variations of the VTA example, where the number of the participating processes grows from two up to seven processes. We remark that the VTA process also grows since it interacts with increasing number of services. The ranges of the domain types used in the messages (e.g. Flight, Time) were set to three values for each type. Although the examples described in the paper are relatively simple, they still are considerably more complex with respect to the samples presented in other tools (e.g., [7, 6]).

In order to compare the verification complexity on the same scenarios under different communication models, we have used domains where the synchronous model is adequate. We used two properties in the experiments specified as Linear-time Temporal Logic (LTL) formula. The first property (P1) requires that the user process terminates successfully only if also the reservation services do. This property is expected to be valid in the domain, i.e., to be respected by all the executions of the Web service composition. The second property (P2) expresses the possibility for the partners to terminate successfully. This property is expected to be satisfiable, i.e., there are some executions of the composition where the property is true. Moreover, we expect that the verification task produces a trace corresponding to one these executions, thus witnessing the validity of this possibility.

The results of the verification of these properties are summarized in Table 1. The verification was performed using two state of the art model checkers, namely NuSMV [4] and SPIN [9]. We tested the specifications of the composition under synchronous product (Sync), under locally order model (LO) and under the most general model (UO). The table contains information on the time used for the verification and counterexample generation in seconds, and on the size of the state vector in bytes. That is, if the state vector size is 14 bytes, the state space is $2^{8\times 14}$ states.

Some comments on the difference in the performance of NuSMV between the two properties. This is due to the fact that the second property requires the generation of a witness scenario, and this takes a lot of time. The time required by NuSMV to report that the second property can be satisfied without extracting the witness trace is similar to the time required to verify the first property. On the contrary, the verification using SPIN model checker requires much more time for the first property. Indeed, in this case all the behaviors have to be considered to prove the correctness of the property, while a single witness trace is sufficient for the second property.

The presented results demonstrate the reduction of the verification performance when more general communication model is applied. This is explained by the fact that more general model introduces more queue variables and therefore increases the state space size. This is particularly important for the NuSMV model checker, since the techniques used there strictly depend on the number of variables, their domains and relations.

7. RELATED WORK AND CONCLUSIONS

In this paper we presented a unified framework for the analysis and verification of Web service compositions provided as BPEL specifications. The framework is based on a definition of composition of BPEL processes that is parametric with respect to the communication model. The class of compositions that we are able to model and analyze is substantially larger than those covered by other verification approaches. We also provide an algorithm that is able to identify the simplest communication model that is adequate for the composition, and to build the corresponding composition. Our experiments show that choosing the right model can lead to a substantial improvement in the verification performance.

In our earlier work [11], we made an initial attempt to design a framework suitable for representing different communication mechanism. The framework was based on a specific form of composition, referred as *extended parallel product*, and was not able to handle different models in a uniform way and to define relations between them. As a result, that approach lacks the completeness and expressiveness of the formalism presented in this paper. Also the experimental

Table 1: Verification results

Property	N	NuSMV						SPIN					
		Sync		LO		UO		Sync		LO		UO	
		Time	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size
P1	2	0.08	5	0.09	6	0.09	6	0.87	48	0.89	56	0.89	76
	3	0.11	8	0.13	9	0.15	10	1.05	60	1.09	76	1.13	120
	4	0.19	11	0.22	13	0.24	15	1.51	72	1.60	100	1.75	172
	5	0.26	14	0.32	17	0.36	20	3.67	84	4.79	120	6.37	220
	6	0.32	17	0.42	21	0.50	24	90	100	> 1Gb		> 1Gb	
	7	0.44	20	0.64	24	3.75	27	> 1Gb		> 1Gb		> 1Gb	
P2	2	0.01	5	0.02	6	0.02	6	0.87	48	0.89	56	0.92	76
	3	0.33	8	0.48	9	0.52	10	1.05	60	1.09	76	1.13	120
	4	1.14	11	2.28	13	8.35	15	1.38	72	1.46	100	1.52	172
	5	20.1	14	36.8	17	69.0	20	1.85	84	1.85	120	1.95	220
	6	114	17	221	21	469	24	2.31	100	2.33	140	2.46	272
	7	771	20	1279	24	> 1 hour		2.89	112	2.98	160	3.27	320

evaluation in [11] is more limited and only considers the NuSMV model checker.

In [7] an approach for the analysis of interacting BPEL processes is presented. The approach allows performing a synchronizability analysis of the composition, that is to verify that the synchronous composition may be applied for further analysis without loosing behaviors. However, this approach fails to determine an appropriate model if the composition does not pass that check. On the contrary, our framework extends this approach by providing a complete way to determine such a model and allows for defining of wider class of verification properties.

In [6], process algebras are exploited to verify BPEL processes. More precisely, that approach allows for the analysis of basic properties of BPEL specifications, such as safety and progress checks. The approach is based on the synchronous communications model and therefore is very restrictive with respect to the set of systems it is able to analyze correctly.

In general, the problem of analysis of communication systems with (potentially infinite) channels is widely studied in literature. In spite of certain undecidability results [3], there are a lot of works on restricted subclasses of such systems for which some problems were shown to be decidable (see, e.g., [1]), ranging from synchronous composition to Petri Net-based approaches [16]. While these approaches are devoted to the analysis of a specific communication model in a general setting, our goal is to provide a set of models that allow for a parametric verification in the specific case of Web service compositions.

8. REFERENCES

[1] P. A. Abdulla and B. Jonsson. Channel representations in protocol verification. In *CONCUR*, pages 1–15, 2001.

[2] T. Andrews, F. Curbera, H. Dolakia, J. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weeravarana. Business Process Execution Language for Web Services (version 1.1), 2003.

[3] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[4] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella. NuSMV 2: An OpenSource tool for symbolic model checking. In *Proc. CAV'02*, 2002.

[5] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Elsevier, 1990.

[6] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of Web Service Compositions. In *Proc. ASE'03*, 2003.

[7] X. Fu, T. Bultan, and J. Su. Analysis of Interacting BPEL Web Services. In *Proc. WWW'04*, 2004.

[8] S. Graham, S. Simenov, T. Boubez, G. Daniels, D. Davis, Y. Nakamura, and R. Neyama. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI*. Sams, 2001.

[9] G. J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.

[10] R. M. Karp and R. E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.

[11] R. Kazhamiakin and M. Pistore. A Parametric Communication Model for the Verification of BPEL4WS Compositions. In *Proc. WS-FM'05*, 2005.

[12] R. Khalaf, N. Mukhi, and S. Weeravarana. Service Oriented Composition in BPEL4WS. In *Proc. WWW2004*, 2004.

[13] J. Koehler and B. Srivastava. Web service composition: Current solutions and open problems. In *Proc. of ICAPS'03 Workshop on Planning for Web Services*, 2002.

[14] S. Nakajima. Model-checking verification for reliable web service. In *Proc. OOPSLA'02 Workshop on OOWS*, 2002.

[15] S. Narayanan and S. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proc. WWW2002*, 2002.

[16] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[17] M. Pistore, M. Roveri, and P. Busetta. Requirements-Driven Verification of Web Services. In *Proc. WS-FM'04*, 2004.