Timed Modeling of Web Services Composition for Automatic Testing

Mounir Lallali *, Fatiha Zaidi †‡, Ana Cavalli *
* Institut National des Télécommunications, INT - CNRS SAMOVAR
9 rue Charles Fourrier, F-91011 Evry, France.
Email: {Mounir.Lallali, Ana.Cavalli@int-edu.eu}
† Univ Paris-Sud, LRI, UMR 8623, Orsay F-91405;
‡ CNRS, Orsay, F-91405.
Email: Fatiha.Zaidi@lri.fr

Abstract

This paper presents the modeling of BPEL (timed) constructs by using a new formalism WS-TEFSM (Web Service Timed Extended Finite State Machine). A formal mapping of all BPEL constructs is proposed as well as a model that corresponds to the BPEL Web services composition. The WS-TEFSM formalism allows to deal with timing constraints, data variables, clocks and priority on transitions. To perform the transformation, we define a renaming function and an asynchronous product of all partial machine corresponding to the BPEL process sub-activities. This model is enriched by the addition of priorities on transitions, which permit to handle the termination of the BPEL process and its sub-activities, and by global variables, which are used in the management of events and faults. This transformation step is essential to ensure the test of Web services. A rigorous approach is crucial as we have to deal with complex systems that manage distribution, low-coupled nature and asynchronous behaviors.

1. Introduction

The rapid growth of Internet users has contributed to promote the Web as a new commercial channel for companies. Web services offer a lot of advantages for companies such as to publish their existing products, to integrate Web services developed by other companies, to develop new Web services for their own needs. The Web services describe a standardized way of integrating Web-based applications using the XML, SOAP, WSDL and UDDI open standards over an internet protocol backbone. To standardize the specification of a Web services composition, IBM and other companies have proposed a language called Business Process Execution Language BPEL [18] that became later an OA-SIS Standard. BPEL is a coordination and composition language that captures business interactions between Web services. It can also be viewed as a workflow language for Web Services. In the area of Web services, the design is the most important phase for the orchestration layer that describes all the behaviors of the services. The growing use of these Web services led to ensure a correct behavior of such services. These last years, the software testing community has started to get involved in the Web services domain.

This paper is focused on the formalism that we propose, i.e., a Web Service Timed Extended Finite State Machine (WS-TEFSM). This formalism is closely related to timed automaton [1] and permits to carry out data variables, timing constraints, clocks, state invariants on clocks and priorities on transitions. We give a detailed semantics of this formal model. We define formally a renaming function of partial machines (taking into account invariants states and priorities on transitions), and an asynchronous product (and its semantics) to handle the complete machine of all the involved sub-activities and elements modeled by partial machines (i.e. partial WS-TEFSM). We detail how we transform the BPEL process and its (timed) constructs (e.g. empty, wait and onAlarm). We define how to model the fault handler and the termination activity of the BPEL process and its constructs. We have enriched WS-TEFSM by some defined variables to throw the fault, to model the correlation set and to handle the BPEL process termination. We have extended the WS-TEFSM with priorities on transitions that permit to block and interrupt the BPEL activities (when a fault is thrown). We will also use our formal model to handle the compensation of the BPEL process and its sub-activities. We note that this formal model can be used to model not only BPEL Web services composition, but Web services choreography and real-time systems.

The paper is organized as follows. We present in section 2 the related work. Section 3 describes the WS-TEFSM
model. In section 4, we detail the mapping of the BPEL constructs to the WS-TEFSM model. Section 5 presents an overview of the automatic testing method of Web services composition. Finally, section 6 concludes the paper and we present future work.

2. Related Work

Currently, in the literature, we can find several papers dealing with BPEL specification and formal models (e.g. automata, Petri nets, and process algebras) of Web services. [15] presents a survey of existing proposals for Web services composition and [22] provides an overview of the different models of BPEL that have been proposed. [20] presents a complete pattern-based Petri net semantics for BPEL. Existing Process algebras (LOTOS [4] for instance) and also new process algebras (for instance Finite State Process notation [10]) have been used to model BPEL. We do not consider these formalisms in our work because we wanted to use our experience on formal methods (as in [5]) and tools in the area of testing and to try to adapt it to the specific domain of Web services. Moreover, most of these works are especially focused on properties verification not on testing. In addition, since state machines are powerful enough to specify the behaviors of web services and are suitable to automated testing, the temporal behaviors are specified by using the WS-TEFSM machines. In [16], the authors specified timed systems by Timed Finite State Machine (TFSM) and proposed a formal method (based on this formalism) to provide good time values to test timed systems (optimal timed test suites). The tool BPEL2STS proposed in [19] translates just a subset of BPEL process in state transition system. In [9], a BPEL process is transformed into PROMELA (the input language of SPIN) and used by the model checker SPIN [11] to generate test suite specifications for BPEL.

In [23] a transformation of BPEL into annotated deterministic finite state automaton is proposed. This formalism does not allow to capture the timing aspects of some BPEL activities and also they do not consider BPEL variables, event and fault handlers. In [17], they propose another formalism that deals with data variables, the extended finite state automata, but no timing constraints are considered. In [13] a formalism taking into account timing constraints is proposed, the WSTTS. Nevertheless, this formalism uses only clocks but no data variables. [7] proposes a common semantic model for timed automata (based modeling languages e.g. UPPAAL [21], IF toolset [12]) called Timed Automata Semantic Unit. This timed automata uses a priority on transitions and urgency degrees to model the time progress. But in our model, we use the state invariants to model the time progress, and the priority on transitions to interrupt the activities machine (if a fault is thrown) and to terminate all the BPEL process. The paper [8] describes an extension of timed finite state automaton with priorities and without variables. With the semantics of this automata, a delay transition can never be blocked, and no transition can be blocked by a delay transition. But, in our WS-TEFSM semantics, delay transitions can be blocked. Finally, we are inspired by [7,13] ([13] to define the asynchronous product of partial WS-TEFSM and [7,13] to define the WS-TEFSM semantics).

3. BPEL Modeling with WS-TEFSM

This section presents the WS-TEFSM formalism, that we have defined to obtain a timed model of a BPEL Web services composition. We have adapted the EFSSM, the Timed Automaton [3,21] and the Web Service Timed Transition System model [13] to capture the specific aspects of the Web services domain. We present the BPEL modeling and the required formal definitions of WS-TEFSM, partial WS-TEFSM and transition priority. We use the WS-TEFSM formalism to model the process element (i.e. the root element of the BPEL document) and a partial WS-TEFSM to model the sub-activities of this element (cf. section 4).

3.1. Timed Modeling Approach of the BPEL

The behavior of a Web service is described by sequences of activities, their execution time and their semantics. For instance, the wait activity is fired when the timeout occurs, the reply of invocation may be considered as an instantaneous activity, while the receive activity may require an arbitrary amount of time. To model this temporal behavior, we propose the WS-TEFSM (cf. Def. 1) which extends EFSSM with clock variables, state invariants on clocks and priorities on transitions. The time progresses in states, and transitions take zero time to be executed. In order to represent the time progress, a local clock is added in the WS-TEFSM. It can be initialized at the beginning of each activity to be executed and can be reset at its end. Moreover, in order to model an absolute time, a global clock (denoted gc) is added in the WS-TEFSM. It can be explicitly set to a certain value at the beginning of the BPEL process execution, and is never reset later. In order to control the time progress (i.e. the waiting time in states), we use a time invariants in states (i.e. explicit time-progress conditions).

All BPEL activities are modeled as instantaneous activities except receive, empty (with duration attribute), wait (with for or until attribute) and onAlarm that are related explicitly to a time notion. These instantaneous activities are modeled, as in [13], as instant transitions called action transitions (cf. Def. 6). These transitions are equivalent to
assign the time invariant \( c \leq 0 \) (where \( c \) is a local clock initialized to zero) to the source state of the transition. In that case, the time cannot progress in this source state. Contrarily, the non instantaneous activities are modeled as delay transitions (cf. Def. 6) that will take place in the right moment. They take certain amount of time represented by time increment in the state and followed by their immediate execution. It is semantically equivalent to add a clock invariant to the source state of the transition depending of time, and guards on the clock variables to the transition guards on BPEL process variables.

For instance, the \( \text{wait} \) activity and the \( \text{onAlarm} \) element are used to represent timeouts. These constructs have two forms. In the first form (with \( \text{for attribute} \) they are fired when certain time (i.e. \( \text{duration} \) \( d \)) has passed. We associate in this case the invariant \( c \leq d \) to the source state of the transition. In the second form (with \( \text{until attribute} \)), the transitions are fired if the current absolute time has the specified value (i.e. \( \text{deadline} \) \( dl \)). We associate the invariant \( gc \leq dl \) to the source state of the transition.

The empty activity (with duration constraints) is equivalent to the sequence of two transitions as in [13]. The first transition is an instant transition that resets the local clock \( c \). The second transition has the clock guard that is evaluated to \( \text{true} \), if the value of the clock \( c \) satisfies the duration constraint (e.g. \( c \leq \text{d} \)). A time invariant \( c \leq \text{d} \) (analogous to the duration constraint) is associated to the intermediate state.

Transition priority is used to model interruptions in real-time systems. In our model, we use priority on transitions to model the fault handlers and the termination of the BPEL process and its sub-activities (cf. section 4.6). Each non atomic activity can be interrupted by adding an urgent transition (called stop transition from each state to a particular stop state). All the transitions of each BPEL atomic activity have a highest priority and cannot be blocked or interrupted. A delay transition has a lowest priority.

3.2. Timed Extended Finite State Machine with Priorities (WS-TEFSM) Model for Web Service

We introduce in this section the formal definition and semantics of the WS-TEFSM. It is an extended finite state machine enriched with a set of clock variables and priorities on transitions. The values of these clock variables increase with the time progress. In our model, the states of the WS-TEFSM are defined with state invariants that express simple clocks conditions. These invariants should be \( \text{true} \) when the system is in the state.

**Definition 1 (WS-TEFSM)** A machine WS-TEFSM \( M \) is a tuple \( M = (Q, \Sigma, V, C, q_0, F, T, \text{Pri}, \text{Inv}) \) where:

- \( Q = q_0, q_1, ..., q_n \): A finite set of states;
- \( \Sigma = a, b, c, ..., \): Alphabet of the actions including symbols \( P^m \) (output action) and \( P^m \) (input action);
- \( V \): Finite set of data variables where \( \vec{v} = (v_1, v_2, ..., v_m) \);
- \( C \): Finite set of clocks where \( \vec{c} = (c_1, c_2, ..., c_n) \);
- \( q_0 \in Q \): Initial state; \( F \subseteq Q \): Set of final states;
- \( T \subseteq Q \times A \times 2^C \): Transition relation that:
  - \( P \in Q \times \Sigma \times P(V) \land \phi(C) \times \mu \times 2^C \) where:
  - \( P(\vec{v}) \land \phi(\vec{c}) \): Guard condition is logical formula on data variables and clocks;
  - \( \mu(\vec{v}) \): Data variables update function;
  - \( 2^C \): Set of clocks to be reset.
- \( \text{Pri} : T \times D(C) \to N \geq 0 \) assigns to each transition its priority that respects the clock valuation \( u \);
- \( \text{Inv} : Q \to \Phi(C) \) assigns a set of time invariants (logical formulas) to the states.

### 3.2.1 Data Variables

We consider the basic data domains (\( \mathbb{R} \) of reals, \( \mathbb{Z} \) of integer and \( \mathbb{B} \) of booleans) and the structured domains (range, array and record). We note \( D_V \) the universal data domain used to abstract data variables. A variable in \( D_V \) can hold data of any type. A logical constraint on data variables is noted \( P(V) \). The update of data variables represents the update action \( \vec{v} := \mu(\vec{v}) \). It is denoted by \( [\vec{v}] := x \).

**Definition 2 (Data Variables Valuation)** Let \( V \) a finite set of data variables. A valuation \( v \) over \( V \) is a function \( v : V \to D_V \) that assigns to each variable \( x \in V \) a value in the data variables domain \( D_V \). The initial data variables valuation is noted \( v_0 \).

\[ v[\vec{v}] := x \] denotes the data valuation which updates the variables \( \vec{v} = (v_1, ..., v_m) \) and leaves the rest of variables (i.e. \( V \setminus \{v_1, ..., v_m\} \)) unchanged.

### 3.2.2 Clock Variables

The clocks are divided in two types: local clocks noted \( c_i \) and global clock noted \( gc \). We assume that the clocks values \( c_i \) can be compared to the non-negative real numbers (i.e. \( c_i \in \mathbb{R}_+ \)) or reset to 0. The \( gc \) clock expresses the global time and can not be reset to 0. But, their values can be compared to the non-negative reals numbers. The clocks update consists in the reset of \( R \subseteq C \) of clocks. It is denoted by \( R \mapsto 0 \).

**Definition 3 (Clock Valuation)** A clock valuation \( u \) over the set of clocks \( C \) is a function \( u : C \to D_C \) (noted by \( u \in D_C \)) that assigns to each clock \( c \in C \) a value in \( R_+ \). The initial clock valuation \( u_0 \) corresponds to the initialization to 0 of all clock variables:

\[ \forall c \in C, u_0(c) = 0. \]
3.2.3 State Invariants

In our model, the states of the WS-TEFSM are associated with state invariants (logical formulas on clock values) that express simple clocks conditions. The state invariants should be true when the machine is in the state. This machine may remain in a state as long as the clock valuation satisfies the invariant condition of that state. We can assign a set of time invariants to one state because it can be the source state of several transitions such as switch and pick activities. This set associated to a state q is denoted by Inv(q) = \{e₁, e₂, ...\}. We will write u ∈ Inv(q) to denote that the clock valuation u satisfy eᵢ | ∃eⱼ ∈ Inv(q). We define also the conjunction of set invariants that are used in the definition of the asynchronous product (cf. Def. 11,12) by: Inv(qᵢ) ∨ Inv(qⱼ) = \{eᵢ ∨ eⱼ|∀eᵢ, eⱼ ∈ Inv(qᵢ), ∀eⱼ ∈ Inv(qⱼ)\}.

For more details of data variables constraints, clocks constraints and state invariants, interested readers can refer to the full version of this paper available at [14].

3.2.4 Transitions

The transitions are annotated with the set of guards, actions, data variable updates and clock resets.

Definition 4 (Transition) Each transition t = qᵢ → qⱼ is associated to a triple l = cond, a, [v := x; R] where:

- cond is a guard on data and clocks;
- a ∈ Σ is the action symbol;
- v := x is the data variables update;
- R is the clocks set to be reset.

The actions in Σ represent an observable actions. The label ε ⊈ Σ denotes an internal action that is unobservable. We note Σε the set Σ \{ ε \}.

3.2.5 Transition Priority

The transition priority depends to time and can be dynamically updated with respect to time progress. In our model, an enabled transition can block another if it has a higher priority.

Definition 5 (Transition Priority) A transition priority is a function Pri : T × R⁺ \[ T \rightarrow \mathbb{N}_≥0 \] that assigns a non-negative integer value Pri(t, u) to each transition t ∈ T its priority with respect to a clock valuation u ∈ R⁺.

As in [7], the priority has three levels: (i) The highest priority (noted by p) enables to model atomic actions that cannot be interrupted, (ii) The lowest priority (noted by l_p) is the time progress priority, (iii) The middle (called urgent) priority has a set of levels. All urgent priority transitions block time progress. In our model, we distinguish two urgency levels: stop s_p and basic b_p where b_p < s_p. All instant transitions (except the transitions of the atomic activities) have a basic priority b_p. The level s_p is used to interrupt and block all the enabled transitions in the process or scope termination (cf. section 4.6).

3.2.6 Semantics of WS-TEFSM

The WS-TEFSM semantics is defined, as in [7, 13], by a labeled transition system (LTS). The delay transition [3] indicates that if the others transitions going out from the same source state have a lower priority, then the machine does not execute any action. In other words, the machine does not change state, but increments the current value of the clocks d by u ⊕ d (that represents a valuation where all clocks have advanced by the real d from their value in u). A delay transition may affect the priority of other transitions through the function Pri and not blocks any transition. The priority of the delay transition is a constant value zero (i.e. the value of the lowest priority l_p). The action transition indicates that transitions out from the source state of this transition have a highest priority, and if the condition cond is valid, then the machine follows the transition by executing the action a, changing the current values of the data variables by the action [v := x] (i.e. v = v′ := [v := x]), resetting the subset clocks R (the clock valuation u resets each clock in the set R) and moving in the next state q′. Particularly in the case of the internal action ε, the clocks remain unchanged and consequently u′ = u.

In WS-TEFSM semantics, no transition can be blocked by a delay transition which, on the contrary of [8], can be blocked.

Definition 6 (WS-TEFSM Semantics) Let M be a WS-TEFSM that M = (Q, Σ, V, C, q₀, F, T, Pri, Inv). The WS-TEFSM semantics is defined by a labeled transition system Sem_M = (S, s₀, Γ, →):

- S ⊆ Q × R⁺ \[ C \times D_V^{|} \] is the set of semantic states (q, u, v)
- q is a state of a machine M;
- u is an assignment (i.e. clock values represented by clock valuation u) that satisfies one invariant of the state q (i.e. u ∈ Inv(q));
- v is a data values represented by data variable valuation v.
- s₀ = (q₀, u₀, v₀) is the initial state;
- Γ = Σε ∪ {d | d ∈ R⁺} is the labels set where d corresponds to the elapse of time.
- → ⊆ S × Γ × S is the transition relation defined by:
  - action transition: Let (q, u, v) and (q′, u′, v′) be two states. Then (q, u, v) → a (q′, u′, v′) if
    ∃t = q → cond.a.[v := x; R] → q′ ∈ T such that
    * u ∈ cond.a, u′ = u[R ⊕ 0], u′ ∈ Inv(q′),
    * v′ = v[v := x]
  - delay transition: Let (q, u, v) and (q′, u′, v′) be two states. Then (q, u, v) → d (q′, u′, v′) if
    ∃t = q → d cond.a.[v := x; R] → q′ ∈ T such that
    * u ∈ cond.a, u′ = u[R ⊕ 0], u′ ∈ Inv(q′),
    * v′ = v[v := x]
  - internal transition: Let (q, u, v) and (q′, u′, v′) be two states. Then (q, u, v) → ε (q′, u′, v′) if
    ∃t = q → ε cond(u, v) → q′ ∈ T such that
    * u ∈ cond(u, v), u′ = u[R ⊕ 0], u′ ∈ Inv(q′),
    * v′ = v[v := x]
3.3. Partial WS-TEFSM

This section introduces the formal definition of the partial WS-TEFSM and the associated renaming function (inspired by [23]). We propose an asynchronous product of these partial machines which are used to model structured activities by combining their sub-activities.

Definition 7 (Partial WS-TEFSM) A partial WS-TEFSM \( PM = (Q, \Sigma, V, C, \ell, Q_{in}, Q_{out}, F, T, Pri, Inv) \) is an WS-TEFSM extended by input state \( q_{in} \) (representing the entrance state of the partial machine and which replaces the initial state \( q_0 \)) and a \( Q_{out} \) set of output state \( q_{out} \) (representing the exit state of the partial machine).

3.3.1 Renaming State and Partial WS-TEFSM Functions

The Renaming Function \( \overline{\sigma} \) of State allows renaming a state and updating its invariants. The renaming function of Partial WS-TEFSM \( \sigma \) that uses the function \( \overline{\sigma} \) permits to rename a state of a partial WS-TEFSM.

Definition 8 (Renaming State Function) Let \( \overline{q} \) and \( q' \) be two states of two partial WS-TEFSM machines. The function \( \overline{\sigma} \)
defined on a state \( \overline{q} \) renames a state \( q \) in a state \( q' \) and is defined by:

\[
\overline{\sigma}(\overline{q}, q \rightarrow q') = \begin{cases} q' & \text{if } \overline{q} = q \\ \overline{q} & \text{otherwise} \end{cases}
\]

We note \( \overline{\sigma}_T \) the renaming transition function where:

\[\forall t \in (q_{in}, l, q_{end}) \in T, \overline{\sigma}_T(t, q \rightarrow q') = ((q_{in}, q \rightarrow q'), l, (q_{end}, q \rightarrow q'))\].

Definition 9 (Renaming Partial WS-TEFSM Function) Let \( PM = (Q, \Sigma, V, C, q_{in}, Q_{out}, F, T, Pri, Inv) \) be a partial WS-TEFSM. The function \( \sigma \) allows to rename PM in another partial WS-TEFSM \( PM' = (Q', \Sigma, V, C, q_{in}', Q_{out}', F', T', Pri', Inv') \) such that:

- \( Q' = \{ q' \} \cup (Q \setminus \{ q \}), q_{in}' = \overline{\sigma}(q_{in}, q \rightarrow q') \);
- \( Q_{out}' = \bigcup_{q' \in Q_{out}} \overline{\sigma}(q', q \rightarrow q') \);
- \( T' = \{ \overline{\sigma}_T(t, q \rightarrow q') \mid t \in T \} \);
- \( Pri' = \bigcup_{t \in T} \overline{\sigma}_T(t, q \rightarrow q'), u) \);
- \( Inv' = \bigcup_{(q', e) \in Inv} \begin{cases} (q', e' \land \overline{\sigma}(q, e \rightarrow e')) & \text{if } \overline{q} = q \\ \emptyset & \text{if } \overline{q} = q_{in} \\ (q, \overline{e}) & \text{otherwise} \end{cases} \).

Let \( \overline{Q} = \{ q_1, ... , q_n \} \) be a finite set of states. We define \( \sigma' (PM, \overline{Q} \rightarrow q') = \sigma((PM, q_1 \rightarrow q'), ..., q_n \rightarrow q') \).

3.3.2 Asynchronous Product of Partial WS-TEFSMs

To formalize the parallel execution of the concurrent activities (e.g. flow activity), we define the asynchronous product (inspired by [13]) \( PM_1 \otimes PM_2 \) of two partial WS-TEFSMs \( PM_1 \) and \( PM_2 \). This product is generalized to \( n \) machines and is denoted by \( \prod_{i=1}^n PM = PM_1 \otimes \cdots \otimes PM_n \).

In the following we consider that \( PM_1 \) and \( PM_2 \) are two partial WS-TEFSMs, \( D_T \) (i.e., the universal domain of the data variables of \( PM_1 \) and \( PM_2 \)) are a finite set such that: \( \forall \ell \in [1..2] PM_{\ell} = (Q_{\ell}, \Sigma, V, C, \ell_{in}, \ell_{out}, F_{\ell}, T_{\ell}, Pri_{\ell}, Inv_{\ell}) \).

Definition 10 (Synchronization Function) Let \( f_T \subseteq \Sigma \times \Sigma \) be a function: \( (\ell_{in}, \ell_{in}) \in f_T \Rightarrow (\ell_{in}, \ell_{in}) \in f_T \forall \ell_T \in \Sigma \).

Definition 11 (Asynchronous Product of Partial WS-TEFSMs) The asynchronous product of \( PM_1 \) and \( PM_2 \) is a WS-TEFSM \( PM = PM_1 \otimes PM_2 = (Q, \Sigma, V, C, q_{in, out}, F, T, Pri, Inv) \) where:

- \( Q = Q_1 \times Q_2 \): the states of \( PM \) are formed by the couples of states of each machine. We only consider the accessible states from the initial state by the transition relation \( T \);
- \( \Sigma = \Sigma_1 \cup \Sigma_2 \): The alphabet of actions is the union of the alphabets of the two partial WS-TEFSM;
- \( V = V_1 \cup V_2 \): \( C = C_1 \cup C_2 \); \( F = F_1 \cup F_2 \);
- \( q_{in} = (q_{in, 1}, q_{in, 2}) \): The input state formed by the input state of \( PM_1 \) and \( PM_2 \);
- \( q_{out} = \{ q_{out, 1} \times q_{out, 2} \} \): the set of output state formed by output states of \( PM_1 \) and \( PM_2 \);
- \( T : Q \times A \rightarrow 2^Q, Pri : T \times D_T \rightarrow N_\geq 0 \);
- \( Inv : Q_1 \times Q_2 \rightarrow \Phi(C) \) is defined by:

\[
Inv = \bigcup_{(q_{in, \ell}, \ell_{in}) \in f_T} ((q_{in, \ell}), \ell_{in} \land \ell_{in}) \).
\]

Let \( a_1 =1 \) and \( a_2 =2 \). Let \( l, l_1, l_2 \) and \( l \) be action transitions such that:

- \( l = cond, a_1, [v_1 =X, R] \); \( l \in [1..2] \)
- \( \forall \ell \in [1..2]\); \( \ell_{in} = cond, a_1, [v_1 =X, R] \);
- \( \ell = cond, \land cond_2, (l, m, \ell_m), [v_1 =X_1, v_2 =X_2, \ell_{in} \land \ell_{in}] \).

Definition 12 (Asynchronous Product Semantic) The semantics of \( PM_1 \otimes PM_2 \) is defined by labeled transition system \( (S, s_0, \Gamma, \Rightarrow) \) where:

- \( S = (Q_1 \times Q_2) \times R_{\Sigma}^{\ell_{in}} \times [C_1] \times D_T^{\ell_{in}} \times [C_2] \times [\ell_{in}] \);
- \( s_0 = ((q_{in, 1}, q_{in, 2}), u_{in, 1}, v_{in, 2}) \) is the initial state where:

- \( u_{in, 1} = u_{in, 1} \cup u_{in, 2}, \) the initial clocks assignment of the two partial machines;
- \( v_{in, 1} = v_{in, 1} \cup v_{in, 2}, \) the initial data variables assignment of the two partial machines;
- \( \Gamma = \Sigma_1 \cup \{ d \mid d \in R_+ \} \) is the labels set where \( d \) corresponds to the elapse of time;
4. From BPEL Constructs to WS-TEFSM

In this section, we give the definition of some BPEL constructs (activities and elements) in terms of the WS-TEFSM formalism presented in the previous section. For more details of the modeling of the communication activities, the structural activities, the correlation sets, the event handlers, the links, and the termination of the BPEL constructs (e.g. structural activities, event and fault handlers), all details are given in the complete version of this paper available at [14].

4.1. The Process Element

A BPEL process always starts with the process element (i.e. the root of the BPEL document). It is composed of the following optional children: partnerLinks, partners, variables, correlationSets, faultHandlers, compensationHandlers and eventHandlers. These elements are presented below. The process element contains the actual workflow definition with the top level activity declaration. It is denoted in BPEL as <process name=string> partnerLinks? partners? variables? correlationSets? faultHandlers? compensationHandler? eventHandlers? activity </process>.

The execution of sub-activities is carried out in parallel because the fault, the event and the compensation handlers can be carried out independently of the principal process activity (i.e. structured activity of the process element noted above by activity). If this is not the case, these sub-activities are synchronized by the tuple (sending.reception). This execution can be modeled then by the asynchronous product detailed in Def. 11 and 12.

4.1.1 The Data Variables and the Clocks Set

The data variables set $V$ is partitioned into disjoint sets i.e. $V = V_B \cup V_L \cup \{V_C \cup V_CS \cup V_F \cup \{stopProcess\})$ where:

- $V_B$ is a finite set of BPEL variables defined explicitly by the variables element of BPEL process;
- $V_L$ is a finite set of link variables defined by the partners, partnerLinks and links elements of BPEL process;
- $V_C$ is a finite set of constraint variables;
- $V_CS$ is a finite set of global correlation variables;
- $V_F$ is a finite set of global fault variables;
- stopProcess is a global boolean variable that the true value initiates the termination of all the process.

The real values of the clock variables increase with the passing of time. All clocks progress synchronously [21]. These clock variables are partitioned into two disjoint sets i.e. $C = \{c_i, \forall i\} \cup \{gc\}$ where $C$ is the clock sets, $c_i$ is a local clock and gc a global clock used with the wait until and onAlarm until constructs.

4.1.2 The Process Element WS-TEFSM

The process model is based on the asynchronous product of the partial WS-TEFSMs of the process sub-activities by renaming the tuple of the input states $(q_1, in, ..., q_4, in)$ (respectively the tuple of the output states $(q_1, out, ..., q_4, out)$) of all the sub-activities partial machines by a new input state $q_{in}$ (respectively by a new output state $q_{out}$).

Let $\{PM_i, i \in [1..4]\}$ be the partial machines of the process sub-activities. Then
4.2. Modeling Basic Activities

Basic activities are empty, throw, terminate, assign and wait activities. The assign activity is modeled as in [17] by using the partial WS-TEFSM.

4.2.1 The Empty Activity

The empty activity is used when there is a need of doing nothing [18]. The BPEL syntax of this activity is $<empty duration=d/>$. The empty is modeled as a partial WS-TEFSM $\text{PM} = \{\{q_{in}, q_{out}\}, \emptyset, \emptyset, \{c\}, q_{in}, q_{out}, \emptyset, \{t_1, t_2\}, \text{Pri}, \{\{q_{in}, c \leq 0\}, \{q_{in}, c \cdot d\}, \{\text{true}\}\}\}

\begin{itemize}
  \item $t_1 = (q_{in}, < c \cdot d, [; c] >, q_{out})$
  \item $\text{Pri} = \{(t_1, \_), (t_2, \_), h_{fp}\}$
\end{itemize}

4.2.2 The Wait Activity

The wait activity allows to wait for a given time period (i.e. duration $d$) or until a certain time has passed (i.e. deadline $dl$). Exactly one of the expiration criteria must be specified [18]. The wait syntax is $<\text{wait (for}=d\text{) until}=dl\text{}/>$. Let $\text{Pri}_{w} = \{(t_1, \_), l_{p}\}$.

\begin{itemize}
  \item $<\text{wait for}=d\text{}/>\text{ is modeled as a partial WS-TEFSM}$
  \[\text{PM} = \{\{q_{in}, q_{out}\}, \emptyset, \emptyset, \{c\}, q_{in}, q_{out}, \emptyset, \{t_1\}, \text{Pri}_{w}, \{\{q_{in}, c \leq d\}, \{\text{true}\}\}\}\]
  \[ - t_1 = (q_{in}, < c < d, [; c] >, q_{out})\]
  \item $<\text{wait until}=dl\text{}/>\text{ is modeled as a partial WS-TEFSM}$
  \[\text{PM} = \{\{q_{in}, q_{out}\}, \emptyset, \emptyset, \emptyset, q_{in}, q_{out}, \emptyset, \{t_1\}, \text{Pri}_{w}, \{\{q_{in}, c \leq dl\}, \{\text{true}\}\}\}\]
  \[ - t_1 = (q_{in}, < gc < dl, [; c] >, q_{out})\]
\end{itemize}

4.2.3 The Throw Activity

The throw activity generates an internal fault inside the business process [18]. It has the following BPEL syntax $<\text{throw FaultName}=f\text{}/>$. It is an internal activity and is modeled as a partial WS-TEFSM

\[\text{PM} = \{\{q_{in}, q_{out}\}, \emptyset, \{v_f, \text{stopScope}\}, \{c\}, q_{in}, q_{out}, \emptyset, \{t_1\}, \text{Pri}, \{\{q_{in}, c \leq 0\}, \{\text{true}\}\}\}\]

\begin{itemize}
  \item $t_1 = (q_{in}, < c < 0, [; c] := f, \text{stopScope} := \text{true}; [; c] >, q_{out})$
  \item $\text{Pri} = \{(t_1, \_), h_{fp}\}$
\end{itemize}

A global fault variable $v_f \in V_f$ is used to match the fault with one of the catch activity. The throw activity assigns the fault name $f$ to the variable $v_f$ and true to the boolean variable $\text{stopScope}$.

4.2.4 The Terminate Activity

The terminate activity is used to immediately terminate the behavior of a business process instance [18]. The BPEL syntax of this activity is $<\text{terminate} />$. The terminate activity is modeled as a partial WS-TEFSM

\[\text{PM} = \{\{q_{in}, q_{stop}\}, \emptyset, \{\text{stopProcess}\}, \emptyset, q_{in}, q_{stop}, \emptyset, \{t_1\}, \text{Pri}, \{\{q_{in}, \text{true}\}, \{\text{true}\}\}\}\]

\begin{itemize}
  \item $t_1 = (q_{in}, < c < 0, [; \text{stopProcess} := \text{true}; [; c] >, q_{stop})$
  \item $\text{Pri} = \{(t_1, \_), h_{fp}\}$
\end{itemize}
4.3. Modeling Communication and Structural Activities

Communication activities, such as receive, reply and invoke, are used to exchange messages with partners. Communication Activities except synchronous invoke are represented as a partial WS-TEFSM with a single action transition per exchanged message. Structural activities are sequence, while, switch, flow and pick. We model these communication and structural activities as in [17, 23] by using our formalism (i.e. partial WS-TEFSM). For more details of the modeling of the communication and structural activities, one can refer to the full version of this paper available at [14].

4.4. Modeling Scope Activity

The scope activity is the core construct of data handling, fault handling, and compensation handling in BPEL [18]. Each scope has a primary activity (basic or structured) which defines the normal behavior of the scope. This activity is denoted in BPEL as: <scope> variables? correlationSets? scope? faultHandlers? compensationHandler? eventHandlers? activity </scope>. After an activity throws a fault, the fault handler of the scope finishes the control flow inside the scope first and handles the fault (cf. section 4.5).

Let \( \{ PM_i, i \in [1..5] \} \) be the partial machines of the scope sub-activities. Then

\[
PM = \{ PM_1 = PM_{activity}, PM_2 = PM_{eventHandlers}, PM_3 = PM_{scope}, PM_4 = PM_{faultHandlers}, PM_5 = PM_{compensationHandler} \}.
\]

Let \( PM' \) and \( PM'' \) the machine modeling the control flow and respectively the fault and compensation handlers where:

\[
PM' = \sigma(\{ PM_{i, \in [1..3]} \}_{i=1}^{3}, F', T', Pri', Inv''),
\]

\[
PM'' = \sigma(\{ PM_{i, \in [1..5]} \}_{i=4}^{5}, F'' \cup q_{out} \cup q_{in} \cup q_{out}' \cup q_{in}' \cup q_{out}'').
\]

The scope machine is defined as the process machine (cf. section 4.1). The scope data variables set \( V \) is partitioned into following disjoint sets: \( V = V_B \cup V_C \cup V_S \cup V_F \cup \{ stopScope \} \) such that \( V_B, V_C, V_S, V_F \) and \( stopScope \) are defined above (cf. section 4.1.1), and \( stopScope \) is a global boolean variable that the true value initiates the scope termination (i.e. the termination of all active activities directly enclosed within this scope). A scope correlation sets is defined as the process correlation sets by a subset of correlation variables \( V_S \subset V \).

4.5. Modeling Defined Fault Handlers

The fault handler of a scope or a process is a set of catch clauses defining how the scope should respond to different types of faults. After an activity has thrown a fault, the fault handler of the scope has to finish the control flow inside the scope first. Afterwards it has to handle the fault [18]. The faultHandlers element combines a switch activity applied to various sequences of a catch or a catchAll activities and a sub-activities partial machine. The catchAll element is used to catch all the faults that are not handled by the defined catch activities.

Let \( Pri = \{ (t_i, b_p) \} \). The catch activity is modeled as a partial WS-TEFSM which matches the value of the instantiated fault variable \( v_f \in V_F \) (i.e. the throw fault name):

\[
PM = \{ \{ q_{in}, q_{out} \}, \emptyset, \{ fn, v_f , \{ c \}, q_{in}, q_{out} \}, \emptyset, \{ t_i \}, Pri = \{ (q_{in}, c \leq rand(C)), (q_{out}, true) \} \}
\]

- \( t_i = \{ (q_{in}, c < rand(C) \land fn = v_f , \{ c \}) \land q_{out} \}
- Pri = \{ (t_i, b_p) \}

and the catchAll activity is modeled as a partial WS-TEFSM:

\[
PM = \{ \{ q_{in}, q_{out} \}, \emptyset, \{ fn, v_f , \{ c \}, q_{in}, q_{out} \}, \emptyset, \{ t_i \}, Pri = \{ (q_{in}, c \leq rand(C)), (q_{out}, true) \} \}
\]

- \( t_i = \{ (q_{in}, c < rand(C) \land fn = v_f , \{ c \}) \land q_{out} \}
- Pri = \{ (t_i, b_p) \}

4.6. The Termination Activity

The behavior of a fault handler for a scope begins by implicitly terminating all active activities directly enclosed within this scope. We use two boolean variables (stopProcess and stopScope) and the transition priority to handle the process termination (initiated by the terminate activity) or the scope termination (initiated by the fault handler). stopProcess is a global variable of the process that is assigned to true by the terminate activity. stopScope is a local variable of each scope that is assigned to true by the scope throw activity.

To interrupt a non atomic activity, we add a stop transition from all the states \( q_j \) (except the output state) of the activity machine to the stop state \( q_{stop} \). This transition has the form \( (q_j, stopProcess \lor stopScope, q_{stop}, false) \). It has also the stop priority \( s_p \) (urgent priority higher than lowest \( l_p \) and basic urgent \( b_p \) priorities) and can block each enabled transition from the same source state. We detail below the termination of the BPEL constructs.

We detail below the wait and the BPEL process termination. We give more details about the termination of the structural activities, and the event and fault handlers [14].

4.6.1 Atomic Activities

The termination is not applied to empty, terminate and throw activities which are atomic. All transitions of their partial WS-TEFSM have a highest priority noted \( h_p \).
4.6.2 Termination of Basic and Communication activities

Each wait, receive, reply and invoke activity can be interrupted and terminated prematurely. To terminate a wait activity, for instance, we add a stop transition \( t_2 \) to a wait machine (cf. section 4.2.2) which has a urgent priority and can interrupt the transition \( t_1 \). The wait machine with termination is defined as:

\[
PM = \{ \{ q_{in}, q_{out}, q_{stop} \}, \emptyset, \emptyset, \{ c \}, q_{in}, q_{out}, \emptyset, T, Pri, Inv \}
\]

- \( T = \{ t_1, t_2 \} \)
- \( Pri = \{ (t_1, \_, l_p), (t_2, \_, s_p) \} \)
- \( t_1 = (q_{in}, < c = d, \_, \_[;\{ c \}] >, q_{out}) \)
- \( t_2 = (q_{in}, < cond, \_, \_[;\{ c \}] >, q_{stop}) \)

where \( cond = stop Process \lor stop Scope \).
- \( Inv = \{ (q_{in}, c \leq d), (q_{out}, true), (q_{stop}, true) \} \)

4.6.3 Termination of Process

The process is terminated in one of the following cases [18]: (1) normal termination when the activity describing all the process behavior is completed, (2) abnormal termination when a process is explicitly terminated by a terminate activity or when a fault reaches the process scope, and is either handled or not handled. In the first case, the final state of the process is the output state of the last partial WS-TEFSMs of its sub-activities. In the second case, the termination blocks and interrupts the enabled transitions of all the scope sub-activities of the process. The stop state of the process scope is an accept state.

5. Overview of the Automatic Testing Method of Web Services Composition

This method is based on a referential implementation, i.e., a BPEL Web services composition. We can transform this BPEL Web services composition in a WS-TEFSM by using the mapping rules defined in section 4. Our proposed model WS-TEFSM extends the IF [12] semantics model (i.e. Communicating Extended Timed Automata). Hence, we will describe the Web services composition in IF language according to the WS-TEFSM semantics (see section 3) and the BPEL transformation to WS-TEFSM (see section 4). In addition, we will modify the IF simulator in order to integrate an automated test generation method based on the Hit-Or-Jump algorithm [6] (considered as a new exploration strategy). Finally, a (timed) test case suites will be generated by this new testing tool from the IF specification of the Web services composition. These test cases will be selected to satisfy a coverage criterion (transition coverage, BPEL activity coverage, etc.).

The (timed) test objectives (i.e. the transitions to be covered) will be formulated by a sequence of actions which refers to BPEL process variables, timeouts, activity duration, events and to different BPEL constructs (i.e. activities and elements). In our proposed method, we will use IF observer objects and time requirements to formulate these test objectives. The data and messages contained in the generated test cases will be used to generate SOAP messages in order to test the deployed instance of the composite BPEL Web services. This deployed process is an instance of a BPEL description which differs from the referential implementation of composite BPEL Web services. We use the IF simulator for the following reasons: (i) this tool permits the simulation of the process execution, the management of time, the non-determinism resolution and the representation of the state space, (ii) it can be also extensible in terms of coordination primitives, exploration strategies and heuristics generation as it is an open source tool, (iii) it uses static analysis and partial order reduction to reduce the generated state space. We note that if we have no BPEL implementation and only one informal description of the Web services composition, we can describe directly this process in IF language and use the test generation tool based on IF. The figure 5 presents an overview of the proposed automated testing method of Web services composition.

Figure 1. Overview of the proposed automated testing method

6. Conclusion and Future Work

The main goal of this paper is to provide a WS-TEFSM formalism that represents a BPEL Web services composition. We have proposed a formal model (and its semantics) that deals with data variables, clocks, priorities on transitions, state invariants (that control the time progress) and timing constraints. This paper has also focused on the formal mapping of the BPEL process, its sub-activities (related to time) and elements to our formalism WS-TEFSM. This formal mapping to WS-TEFSM will be used to test the BPEL Web services composition. We advocate that the use of formal methods is essential in the realm of Web services to ensure their correctness and validation. Timing constraints are
of high relevance in this specific domain. Moreover, as we had to deal with different (related time) activities, elements and handlers, we needed: (1) to define a renaming function to glue together all the corresponding partial machines, (2) to define a cross product to deal with asynchronous activities, (3) to define priorities on transitions to interrupt non atomic activities (if a fault is thrown) and to handle the process termination. For sake of simplicity, we have detailed the main BPEL constructs (e.g. scope, event and fault handlers, and termination).

Future work will consist of: (1) proposing the mapping of the compensate activity in WS-TEFSM (2) considering the mapping of referential BPEL process to WS-TEFSM model to describe the Web services composition in IF language, (3) finalizing the IF simulator adaptation to timed test case generation (by using Hit-Or-Jump as a new exploration strategy) and (4) defining an adapted Web services testing architecture. Our formalism is well adapted to handle the needed: (1) to define a renaming function to glue together all the corresponding partial machines, (2) to define a cross product to deal with different (related time) activities, elements and handlers, and termination.

References