Representation, Verification, and Computation of Timed Properties in Web Service Compositions [*]

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

Paritosh Pandya
Tata Institute of Fundamental Research
Homi Bhabha Road, Colaba
Mumbai 400 005, India
pandya@tifr.res.in

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

Abstract

In this paper we address the problem of qualitative and quantitative analysis of timing aspects of Web service compositions defined as a set of BPEL4WS processes. We introduce a formalism, called Web Service Timed State Transition Systems (WSTTS), *to capture the timed behavior of the composite web services. We also exploit an interval temporal logic to express complex timed assumptions and requirements on the system's behavior. Building on top of this formalization, we provide techniques and tools for model-checking BPEL4WS compositions against time-related requirements. We also present a symbolic algorithm that can be used to compute duration bounds of behavioral intervals that satisfy such requirements. We perform a preliminary experimental evaluation of our approach and tools with the help of an e-Government case study.*

1. Introduction

One of the key aspects for the correctness of the Web service compositions is the correctness of the behaviors generated by the interactions among the participants of the composition. In the behavioral analysis of these compositions we require not only the satisfaction of qualitative requirements (e.g. deadlock freeness of the interaction protocols), but also of quantitative properties, such as time, performance, and resource consumption.

Time-related properties are particularly relevant in this setting. Indeed, in many scenarios we expect that a Web service composition satisfies some global timed constraints, and these constraints can be satisfied only if all the services participating to the composition are committed to respect their own local timed constraints. Consider for instance an e-government scenario, where the distributed business process requires the composition of information systems and functionalities provided by different departments or organizations (here, we will consider one of such scenarios, consisting in providing the authorization to open a site for the disposal of dangerous waste). The composite service can comply with the timed commitments with respect to the national regulations (e.g., the duration of document analysis phase) only if they are consistent with the time required by all participating actors to carry out their part of the process.

In the analysis of such properties it is important not only to check whether a certain requirement is satisfied, but also to determine *extremal* time bounds where the property is guaranteed to be satisfied. In e-government scenario, for instance, it is important not only to demonstrate the ability to complete the authorization procedure, but also to find a maximal/minimal duration of such a procedure. However, the computation of these durations by trial and error search of the corresponding values is inherently incomplete and highly inefficient.

In this paper we present an approach for modeling, validating and computing the time-related properties of Web service compositions defined by a set of BPEL processes. We want to stress the fact that the time properties we want to model and analyze are those that are critical from the point of the business logic, i.e., they refer to the time required by the participating actor to carry out their tasks and take their decisions, and to the assumptions and constraints on these times that guarantee a successful execution of the distributed business processes. In e-government scenarios, there times are measured in hours and in days. The "technical" times, which are required, for instance, by the communications among BPEL processes and by the BPEL engines to manage incoming and outgoing message, are orders of magnitude smaller (seconds if not milliseconds) and can be neglected in these scenarios.

[*]This work is partially funded by the MIUR-FIRB project RBNE0195K5, "KLASE", by the MIUR-PRIN 2004 project "STRAP", and by the EU-IST project FP6-016004 "SENSORIA".

This work extends our previous results on the qualitative analysis of time-related properties BPEL compositions [9]. In [9] we define the formalism of *Web Services Timed Transition Systems (WSTTS)*, which allow for modeling the timed behavior of a BPEL composition. WSTTS are closely related to timed automata but incorporate design decisions and features specific to Web service compositions. In [9] we exploit the duration calculus logic for the representation of complex timed requirements in the domain.

In this paper we extend the analysis capabilities of [9] and introduce a decision procedure that can be used to compute extremal durations of intervals satisfying required properties. We adapt Quantified Discrete-time Duration Calculus (QDDC, [12]) to model these properties and adopt the algorithms of [13, 3] to perform the computation in WSTTS models.

The structure of the paper is as follows. In Sect. 2 we introduce the e-government case study that describes the problem of analysis of time-related properties. Section 3 explains how the timed aspects of Web service compositions are modelled in the formalism of WSTTS model, and introduces the Duration Calculus logic exploited to capture the complex quantitative timing requirements and constraints. The qualitative and quantitative timed analysis approach is discussed in Sect. 4 together with some experimental results on the presented case study. Conclusions and future work are presented in Sect. 5.

2. Case Study: e-Government Application

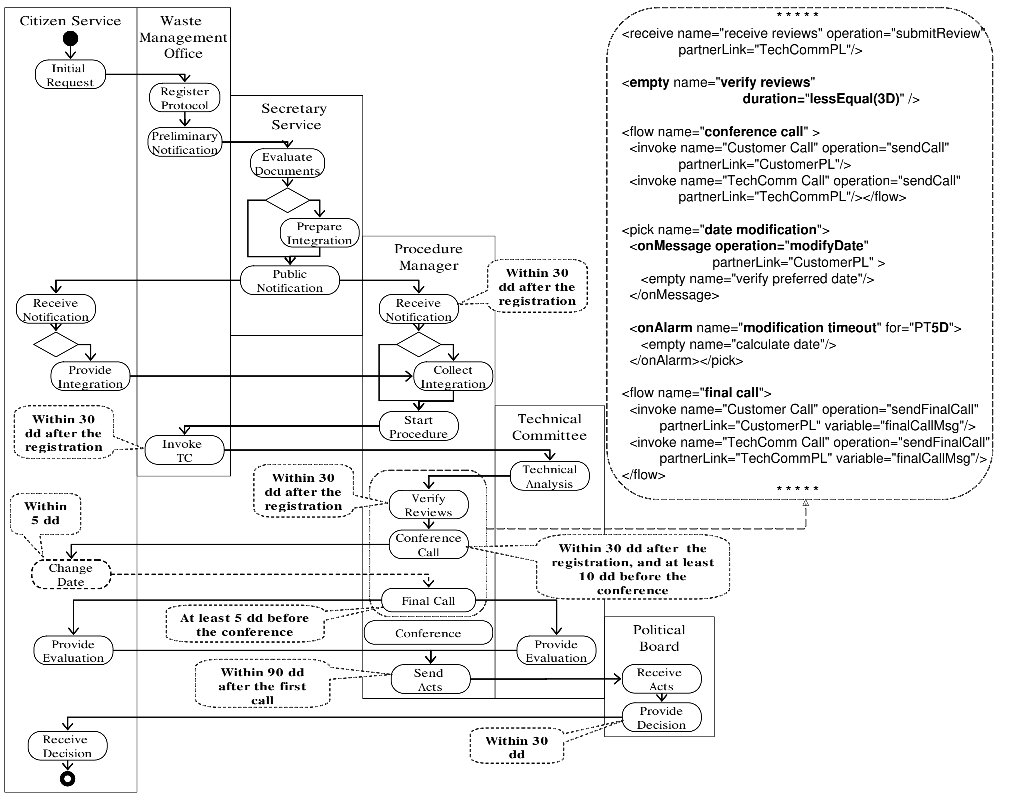
We illustrate our approach with an e-government application. The goal of the application is to provide a service that manages user requests to open sites for the disposal of dangerous waste. According to the existing Italian laws, such a request involves the interaction of different actors of the public administration, namely a Citizen Service, a Waste Management Office (WMO), a Secretary Service, a Procedure Manager, a Technical Committee, and a Political Board. In this application, the whole procedure is implemented as a composition of Web services that serve as interfaces to the processes of the above actors. We model the composition using BPEL specifications to describe the interactions among these actors. The high-level model of the composition is presented in Fig. 1. The procedure describes different phases of the application management where the request is registered, the documentation is evaluated and collected, the application is analyzed regarding the ecological impact of the site, the public conference is scheduled and organized, and final decision is provided.

Apart from the functional requirements, the execution of the process in the choreography should respect a set of timed requirements and constraints, dictated by Italian laws or by the agreement among the involved parties. These requirements (callouts in Fig. 1) specify, for example, that the period of time between the application registration and the notification of the Procedure Manager should not exceed 30 days, or that the participants can change the date within 5 days after the preliminary call. The behavior of the composition and the possibility to satisfy these requirements depend on the time needed for the execution of the activities the involved parties are responsible for. We remark that the critical parameter is the duration of internal activities of the participants, and not to the communication time, which can be neglected.

In these settings, the Web service composition analysis may become a long, error-prone process of finding boundary time values that would ensure correctness of the composition with respect to functional and timed requirements. Consider, for instance, the problem of determining the maximum time which can be spent by the Citizen to provide integration documents and the Technical Committee to perform the analysis, such that the requirement to announce a conference within 30 days after registration is satisfied.

The analysis of time-related aspects of the compositions requires explicit representation of timeouts, operation durations, and even complex properties expressing various timed requirements. While *timeouts* can be represented in BPEL, durations and timed requirements can not, and require specific way to be modelled. In our framework, we assume that the answer times are negligible by default, and that activities that have a non-negligible duration are annotated in the BPEL specification with an extra *duration* attribute. In Fig. 1 an excerpt of the annotated BPEL is presented. Here a BPEL event handler "date modification" is used to model a time-bounded possibility to change the date of the conference. That is, the `onAlarm` activity is triggered if the user does not call the "modifyDate" operation within 5 days. On the contrary, the internal activity "verify reviews" is equipped with a duration annotation to express that certain time may be used for the reviews analysis.

While durations and timeouts can be easily represented within BPEL, *timed requirements* can not and require more powerful notations. Consider, for instance, the requirement that the interval from the registration to the conference call should not exceed 30 days, and it is followed by the interval of length of at least 10 days, ending with the conference. This requirement spans over many activities performed by different parties. In order to be able to handle it, it is necessary to provide a model of the behavior of the BPEL processes that allows for an explicit representation of time. Moreover, it is necessary to exploit techniques for reasoning about time to check if this requirement is satisfied by the BPEL timed model. In the following sections we demonstrate how these issues can be addressed with the help of the WSTTS model, the duration annotations and duration calculus for complex time requirements.



```
*****
<receive name="receive reviews" operation="submitReview"
            partnerLink="TechCommPL"/>

<empty name="verify reviews"
                       duration="lessEqual(3D)" />

<flow name="conference call" >
  <invoke name="Customer Call" operation="sendCall"
            partnerLink="CustomerPL"/>
  <invoke name="TechComm Call" operation="sendCall"
            partnerLink="TechCommPL"/></flow>

<pick name="date modification">
  <onMessage operation="modifyDate"
                  partnerLink="CustomerPL" >
    <empty name="verify preferred date"/>
  </onMessage>

  <onAlarm name="modification timeout" for="PT5D">
    <empty name="calculate date"/>
  </onAlarm></pick>

<flow name="final call">
  <invoke name="Customer Call" operation="sendFinalCall"
        partnerLink="CustomerPL" variable="finalCallMsg"/>
  <invoke name="TechComm Call" operation="sendFinalCall"
        partnerLink="TechCommPL" variable="finalCallMsg"/>
</flow>
*****
```

Figure 1. Waste management application processes

3. Modeling of Time-related Properties

The behavior of a BPEL service is described by sequences of activities. The semantics of these activities and the execution time depend on the type of these activities. For instance, the `onAlarm` activity is fired immediately when the timeout expires. The assignment of variables may be considered as an instantaneous activity, while the service invocation operation may require an arbitrary amount of time since it depends on the time required by the invoked service to fulfill the request. In order to model such behavior, we propose the Web Service Timed Transition System (WSTTS) model, which adopts the formalism of *timed automata* for capturing the aspects specific to the Web service domain. In this formalism, the fact that the operation takes a certain amount of time is represented by time increment in the state, followed by the immediate execution of the operation. In order to guarantee that the transition will take place at the right moment of time, the states and transitions of timed automata are annotated with the invariants and guards of the special clock variables.

Intuitively, WSTTS is a finite-state machine equipped with set of clock variables[1]. The values of these variables increase with the passing of time. A Web service composition thus is represented as a network of several such automata, where all clocks progress synchronously. In this model, the states of the WSTTS are equipped with the *state invariants* that express simple conditions over clocks. The invariants should be true when the system is in the state. Analogously, transitions are annotated with the set of *guards* and *resets*. The former represent simple conditions over clocks, and the latter are used to reset values of certain clocks to zero. The semantic of WSTTS is defined as a labelled transition system, where either the time passes or a transition from one state to another immediately happens.

[1]It is also equipped with the set of non-timed variables of finite domains. For the sake of simplicity, we omit them in the formalism.

Let X be a set of clocks. The constraints on the clock values $\Phi(X)$ are of the form $true \mid x \sim c \mid \phi_1 \wedge \phi_2$, where $\sim \in \{\leq, <, =, \neq, \geq, >\}$, $x \in X$, and $c \in \mathbb{T}$, a domain of time values.

Definition 1 (WSTTS)
A WSTTS is a tuple (S, s_0, A, Tr, Inv), where

- *S is the set of states and s_0 is the initial state;*
- *A is a set of actions; each action is either an input action $?m$, an output action $!m$, or an internal τ action;*
- *$Tr \subseteq S \times A \times \Phi \times 2^X \times S$ is the set of transitions with an action, a guard, and a set of clocks to be reset;*

- $Inv : S \to \Phi(X)$ *assigns invariants to the states.*

In the definition, the effect of the transition $(s, a, \phi, Y, s') \in Tr$ from s to s' is to perform a communication or an internal action a, and to reset a set $Y \subseteq X$ of timers to zero. The transition is possible only if the guard condition ϕ evaluates to true in the source state.

Now we define the semantics of a WSTTS. A clock valuation is a function $u : X \to \mathbb{T}$ from the set of clocks to the domain of time values. Let $\mathbb{T}_C$ denote a set of all clock valuations. Let $u_0(x) = 0$ for all $x \in X$. We will write $u \in Inv(s)$ to denote that u satisfy $Inv(s)$.

Definition 2 (Semantics of WSTTS)
Let (V, S, s_0, A, Tr, Inv) be a WSTTS. The semantics is defined as a labelled transition system $(\Gamma, \gamma_0, \to)$, where $\Gamma \subseteq S \times \mathbb{T}_C$ is a set of configurations, $\gamma_0 = (s_0, u_0)$ is an initial configuration, and $\to \subseteq \Gamma \times \{A \cup tick\} \times \Gamma$ is a transition relation such that:

- $(s, u) \xrightarrow{tick} (s, u + d)$*, if* $(u + d) \in Inv(s)$*, and*
- $(s, u) \xrightarrow{a} (s', u')$*, if there exists* $(s, a, \phi, Y, s') \in Tr$*, such that* $u \in \phi$*,* $u' = u[Y \mapsto 0]$*, and* $u' \in Inv(s')$*.*

That is, either the system remains in the same state and time passes, or a certain transition immediately takes place.

While the WSTTS allows to represent the behavior of a particular service, the behavior of the Web service composition is modelled as a *WSTTS network*. In this model the Web services communicate with each other synchronizing on shared actions, or independently perform their internal activities. The WSTTS network $PP = (X, P_1 \parallel \cdots \parallel P_n)$ consists of n WSTTS P_i over a common set of clocks X. The semantics of the WSTTS network is given in terms of global timed transition system (GTTS). We use $\bar{s} = (s_1, \ldots, s_n)$ to denote a state vector, $\bar{s}_0$ to denote an initial state vector, and $\bar{s}[s_i/s'_i]$ to denote a state vector, where the element s_i is replaced by s'_i.

Definition 3 (GTTS)
Let $(X, P_1 \parallel \cdots \parallel P_n)$ be a WSTTS network. Global timed transition system *has the form* $(\Gamma, \gamma_0, \to)$*, where* $\Gamma \subseteq \langle S_1 \times \cdots \times S_n \rangle \times \mathbb{T}_C$*,* $\gamma_0 = (\langle s_{01}, \ldots, s_{0n} \rangle, u_0)$*, and* $\to \subseteq \Gamma \times \{A \cup tick\} \times \Gamma$ *is a global transition relation defined by:*

- $(\bar{s}, u) \xrightarrow{tick} (\bar{s}, u + d)$*, if* $(u + d) \in \wedge_i Inv_i(s_i)$*;*
- $(\bar{s}, u) \xrightarrow{\tau} (\bar{s}[s_i/s'_i], u')$*, if there exists a transition* $(s_i, \tau, g, Y, s'_i) \in Tr_i$*, s.t.* $u \in g$*,* $u' = u[Y \mapsto 0]$*, and* $u' \in \wedge_i Inv_i(s_i)$*;*
- $(\bar{s}, u) \xrightarrow{m} (\bar{s}[s_i/s'_i, s_j/s'_j], u')$*, if there exist a transition* $(s_i, ?m, g_i, Y_i, s'_i) \in Tr_i$*, and a transition* $(s_j, !m, g_j, Y_j, s'_j) \in Tr_j$*, s.t.* $u \in g_i \wedge g_j$*,* $u' = u[Y_i \cup Y_j \mapsto 0]$*, and* $u' \in \wedge_i Inv(s_i)$*.*

In other words, the model of GTTS allows for three kinds of transitions: a time passing transition, where all clocks increment their values; an internal transition of a particular WSTTS, and a shared communication action of two WSTTS.

3.1. Mapping BPEL Constructs to WSTTS

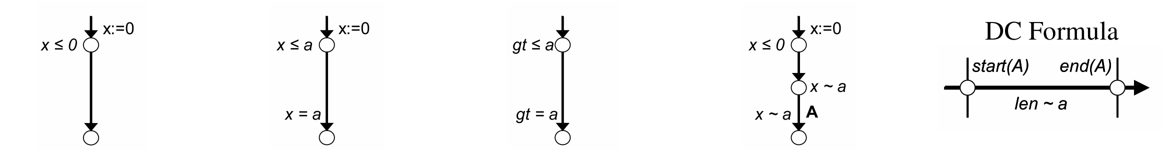
We now give the definition of BPEL constructs in terms of the WSTTS formalism. We remark that, by default, all the activities of the underlying BPEL process are modelled as instantaneous. The fact that a particular activity may have a certain duration is expressed explicitly through *duration annotations* that allow to specify bounds of the activity duration[2].

In this way, until explicitly specified, all internal and message output activities are modelled as instant. Such a transition is semantically equivalent to adding an extra clock `x` to the source state of the transition, and the invariant of the state is `x<=0` (Fig. 2(a)). Hence, time can not pass in the source state of the instant transition. On the contrary, input activities do not require such addition, since they are blocked until corresponding output takes place, and therefore time can pass.

BPEL also defines activities that explicitly refer to time. In particular, the `onAlarm` activity is used to represent timeouts and is modelled as an event handler. This activity has two forms. In the first form (Fig. 2(b)) it is fired when certain time has passed. In the BPEL code represented in Fig. 1 this activity is used to model a timeout of 5 days for the modification of the conference date. In the second form (Fig. 2(c)), `onAlarm` is fired if the current absolute time has the specified value. In order to model the absolute time referenced in this activity, a special clock is added to the WSTTS network model, namely *global_timer*. It can be explicitly set to a certain value in the beginning of the execution, and is never reset later. Another BPEL activity that deals with time is the activity `wait` which blocks the process for a certain time period. Its translation to WSTTS is analogous to that of `onAlarm`.

As we mentioned above, it is possible to constrain the duration of a certain activity. In this way, one is allowed to express simple timed assumptions on the process execution, like service response time, duration of some internal operation or sequence of operations, etc. In this case the activity is explicitly annotated with the duration constraints. This annotation is used in our case study, for example, to denote that the duration of the "verify activity" is less than or equal to 5 days (Fig. 1). Such constraints are conjunctions of the clauses of the form $dur(A) \sim c$,

[2]We stress once more that our goal is to analyze the time properties that are critical for the business logic, and neglect the smaller "technical" times due, e.g. the communications.



instant transition (a) | `onAlarm for="a"` (b) | `onAlarm until="a"` (c) | $dur(A) \sim a$ (atomic activity) (d) | $dur(A) \sim a$ (structured activity) (e)

Figure 2. Time-related constructs as WSTTS

where $\sim \in \{<, >, \leq, \geq, =\}$. When applied to an atomic activity (e.g. `invoke`, `empty`), this annotation is semantically equal to the sequence of two transitions (Fig. 2(d)). The first one is an instant transition and resets the clock `x`. The second transition has the guard that evaluates to true, if the value of the clock `x` satisfies the duration constraints. An analogous constraint is defined in the intermediate state. When the duration annotation is specified for a structured activity, the translation is more complex (Fig. 2(e)). First, all instant subactivities should be converted to non-instant. That is, time can pass arbitrary with these activities. Second, the specification should be constrained to include only the behaviors, in which this complex activity satisfies the duration requirement. This is done by adding to the specification a constraint stating that the time interval from the beginning of the activity ($start(A)$) to its end ($end(A)$) has the required duration. This constraint is specified in the duration calculus, which we will describe in the next section.

Example 1 A part of the WSTTS representing the BPEL code in Fig. 1 is illustrated in Fig. 3. Here the activity "verify reviews", equipped with the duration annotation, is modelled as a sequence of two transitions. The first transition is instant, while the second can be fired within 3 time units. After this activity, the conference call is instantaneously sent to the partners, and a "modifyDate" message is awaited. If the message is received within 5 time units, then a customer-defined date is immediately verified. If the message is not received within 5 time units, the transition corresponding to the `onAlarm` activity is fired, and a final conference date is calculated. In both cases the process instantaneously sends a final conference call to the partners.

3.2. Specifying Time Requirements

We now present a language for specifying complex timed requirements to be verified on the global specification of the composition. Such requirements may express the time intervals between events (or a sequences of events), time bounds on some condition to hold or even complex logical combinations on them. In order to express such properties we exploit a subset of *duration calculus* (DC) [4]. It allows

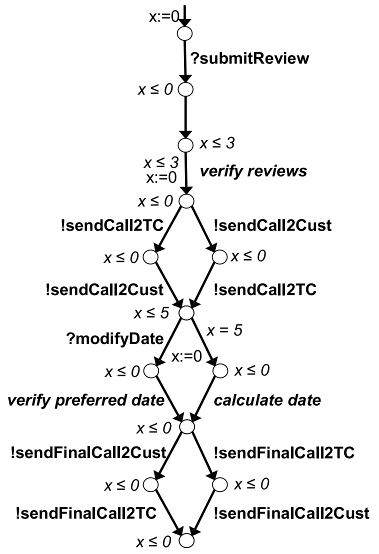


Figure 3. Excerpt of the WSTTS

us to express properties of finite sequences of behaviors and to measure the duration of a given behavioral fragment.

More formally, the logic is defined as follows. Let $Pvar$ be a finite set of propositional variables representing some observable aspects of the composition, and $VAL : Pvar \rightarrow \{true, false\}$ be the set of valuation that assign truth values to each of them. Let also P range over propositional variables, D, D_1, D_2 range over DC formulae, c range over natural number constants, and $\sim \in \{<, \leq, =, >, \geq, \}$. The syntax of the DC formula is:

$$D := \lceil P \rceil^0 \mid \lceil\lceil P \rceil \mid D_1 {}^\frown D_2 \mid D_1 \wedge D_2 \mid \neg D \mid len \sim c$$

DC formulas are evaluated over finite behaviors, i.e., over finite sequences of valuations of propositional variables $VAL(Pvar)^*$.

The constructs above have the following intuitive meaning:

- $\lceil P \rceil^0$ holds for the behavior consisting of a single state satisfying P;

- $\lceil\lceil P\rceil$ requires P to hold at all the states of the behavior (except the last state);

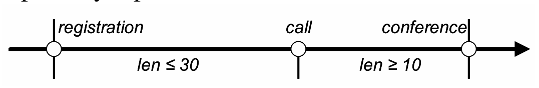
- $D_1 \frown D_2$ splits the behavior into two subintervals, such that D_1 holds for the first subinterval, and D_2 holds for the second one;

- $D_1 \ \wedge \ D_2$ requires both formulas to hold, while $\neg D$ requires that D is not satisfied on the behavior;

- $len \sim c$ relates the duration of the interval with the constant value c (i.e. greater, equal, etc.).

Additionally, we write $\Diamond D = true \frown D \frown true$, if D holds for some subinterval of the behavior; $\Box D = \neg\Diamond\neg D$ denotes, that D holds for all subintervals.

Example 2 Let us consider the requirement that the interval from the protocol registration to the conference call should not exceed 30 days, and that from the call to the conference at least 10 days should pass. The requirement may be graphically represented as follows:



This requirement may be expressed with the following DC formula:

$$\Box(\lceil registration\rceil^0 \frown true \frown \lceil conference\rceil^0 \rightarrow \\ (len \ \leq 30) \frown \lceil call\rceil^0 \frown (len \ \geq 10))$$

The formula says that, for all intervals of the behavior, if the registration happens at the beginning of the interval and the conference at the end, then the interval consists of two intervals with the call in between, such that the duration of the first is less than 30 days, and the duration of the second does not exceed 10 days.

4. Analysis of Time-related Properties

We have implemented the presented ideas as a prototype tool that allows for the qualitative and quantitative timed analysis of Web service compositions. The following elements are being passed to the tool as input:

- composition specification M given as a set of BPEL processes, possibly annotated with the duration constructs;

- set of complex timed constraints C expressed as DC formulas;

- set of target properties P to be verified against the composition;

- set of properties V to be measured using the minimum/maximum algorithms.

Given these inputs, the tool translates them into the specification suitable for the formal techniques, in particular for model checking. The specification reflects the operational semantics of the GTTS given above.

4.1. Discrete Time Representation

In the implementation we adopt a discrete model of time, and use a discrete-time variant of the Duration Calculus, based on (a subset of) Quantified Discrete-time Duration Calculus (QDDC, [12]), to express complex time requirements (an analysis based on a dense model of time under certain conditions may be implemented in a similar way).

The tool performs the transformation of the composition into a finite-state representation. The clock variables are represented as global integer variable that synchronously increment their values when the time elapse transition happens. This event is denoted by a special boolean variable $tick$ that is true when the event is fired. The results of [1] ensure the finiteness of the resulting specification. In particular, it is shown that in the discrete model of time the clock variables may be bounded without affecting the behavior of the system.

4.2. Analysis of Timed Requirements

As we already mentioned, the complex timed requirements may be used in the analysis of the composition in the following ways.

First, the complex time properties may be used to constrain the system behavior. That is, only the executions that satisfy such properties are considered. This is achieved by translating the properties into the corresponding finite-state automata and building a synchronous (i.e. lock-step) product of these automata and a system (as it is done for the duration annotations of the structured activities, Fig. 2(e)). This product i used for further analysis of the composition.

Second, the composition specification can be directly verified against the property. If the property should be satisfied by all the executions of the system (*assertion* property), the tool builds a product of the specification and the automaton corresponding to the negation of the property. This product is not empty when the property is violated. If the property should be satisfied only by some executions of the system (*possibility* property) the tool builds a product of the specification and the automaton of the property. If the product is not empty, then such executions exist. See [9] for more details on these kinds of analysis.

Finally, one can apply quantitative analysis to *measure* a certain property. This analysis allows one to extract time bounds in which the property holds, e.g. the minimal time needed to complete the Waste Management application procedure, or the maximal time of the technical analysis that

```
**min_duration**$(start, final)$
  $R := start \cap reachable(final)$
  $min := 0$
  **if** $(R = \emptyset)$ **return** $\infty$
  **loop**
    $R := immediate(R)$
    **if** $(R \cap final \neq \emptyset)$ **return** $min$
    $R := delayed(R)$
    $min := min + 1$
  **end loop**

**max_duration**$(start, final)$
  $R := start \cap reachable(final) \cap \neg final$
  $max := 0$
  $R' := \emptyset$
  **while**$(R \neq \emptyset \wedge R \not\subseteq R')$ **do**
    $R := immediate(R) \cap \neg final$
    $R' := R' \cup R$
    $R := delayed(R) \cap \neg final$
    $max := max + 1$
  **end while**
  **if** $(R = \emptyset)$ **return** $max$
  **else return** $\infty$
```

Figure 4. Min/Max duration algorithms

still allows to satisfy all the composition requirements. This new kind of analysis is the focus of the next section.

4.3. Quantitative Analysis

Here we present an algorithm that, given a composition specification and a certain property (in QDDC), computes the extremal (i.e. least/greatest) duration of the interval satisfying this property. This algorithm relies on previous results of [13], where the computation of extremal values for *synchronous* systems is addressed, and on the symbolic condition count algorithms of [3].

Intuitively, the algorithm performs as follows. First, the initial system M and a property p are transformed into a system $M'(start, final)$. Minimal (maximal) duration of interval between a state satisfying formula $start$ and a state satisfying formula $final$ in M' is equivalent to the minimal (maximal) duration of interval satisfying p in M. We refer the reader to [13] for the details of this transformation.

Second, the actual extremal value is computed in M' using the algorithms represented in Fig. 4. These algorithms are simple adaptations of the condition count algorithms proposed in [3].

In these algorithms three function are exploited, namely $reachable(R)$, $immediate(R)$, and $delayed(R)$. The first represents a set of states from which the states in R are reachable. The second represents a set of states that are reachable from any state in a set R only through instant transitions. The last represents a set of states that are reachable from any state in R through exactly one $tick$ transition. We remark that in either case we restrict only to the states in $reachable(final)$.

The minimal duration algorithm performs as follows. Initially, the set of states R is a set that satisfies $start$ and from which the state satisfying $final$ is reachable ($reachable(final)$). If the set is empty, there is no interval from $start$ to $final$ and the result of computation is *infinity*. Then we iteratively move from this set of states to their successors until the states satisfying $final$ are reached. When the $tick$ transition is performed, the value of min is incremented.

The maximal duration algorithm is implemented analogously. It progresses from the initial state trying to stay in the states not satisfying $final$. If the is not possible, the current max value is returned. In order to detect an infinite cycle we calculate also a set of states R' that contains all the states visited so far. If the cycle is detected ($R \in R'$), the algorithm returns *infinity*.

4.4. Experimental Results

In order to illustrate the approach represented in the paper, we have conducted a set of experiments on the analysis of the presented case study in different settings. In this experiments we have used the NuSMV model checker [5] for verifying the corresponding finite state automata model. We verified the behavior of the Waste Management Application composition against various properties of interest, and under different assumptions expressed both as the duration annotations and also using QDDC. In particular, we verified the (annotated) composition against the following properties, and experimented with the quantitative analysis:

- Deadlock freeness of the specification;
- Assertion that the procedure always terminates within a given period;
- Possibility to obtain an official conclusion within a given bound;
- Maximal duration of the procedure;
- Minimal time needed to obtain an official conclusion.

The results of the verification are presented in Table 1. The table demonstrates the outcome of the analysis, the verification time, and the size of the reachable state space (i.e., the number of states of the GTTS of the composition specification). Interestingly enough, the computation time is considerably smaller than the verification of the corresponding property, which conforms with the results presented in [13].

Table 1. Experimental results

Property	Result	Time	States
Deadlock	true	0.48 sec	1005
Assertion	false	5,56 sec	2919
Possibility	true	2,28 sec	2919
Max	∞	0.28 sec	1005
Min	10	0.32 sec	1005

5. Conclusions

In this work we presented a formal approach for modeling, verifying and computing of time-related properties on Web service compositions defined by a set of BPEL processes. This approach is based on a formal model, Web Service Timed Transition System, that allows to take into account timed behavior of such compositions. We demonstrated how BPEL time-related constructs can be expressed in this formalism. Moreover, we presented a way to express various time-related requirements and assumptions using both simple modeling constructs or complex DC formulas particularly suitable for expressing such properties. The presented approach enables verification of Web service compositions against time properties using model checking techniques. We also introduced a decision procedure that can be used to compute the time boundaries for intervals satisfying these timed requirements.

Preliminary results on the verification of time-related properties of Web service composition using WSTTS model were discussed in [9]. We extended these results with the techniques for computing quantitative characteristics related to time, and the capabilities to model durations of structured activities.

The problem of the Web service compositions analysis, in particular of BPEL processes, is investigated in the works of [6, 10, 11, 7, 14]. While providing facilities for the verification of processes or their compositions, these approaches do not take time-related properties of composition behaviors into account. The work that is closer to ours is presented in [8]. In this work, a formal model of BPEL processes, μ-BPEL, is presented which allows for mapping from this formalism to a network of timed automata. However, [8] does not provide a way to explicitly specify the transition or state duration bounds, or complex time-related assumptions and requirements as those we model with DC formulas. In [2] temporal abstractions are exploited for the compatibility and replaceability analysis of Web service protocol. In that model one can specify when certain transitions must or may happen, similarly to what we achieve with our duration annotations. However, [2] does not address the problem of the verification of these time properties. Moreover, their temporal abstraction are simple with respect to the set of properties we can express in our approach.

There are several direction for further research. In particular, we are working on the optimizations of the translations from BPEL to NuSMV code and applications of better analysis techniques that give a possibility to drastically improve the verification performance. Another line of research is to replace NuSMV with a model checker, such as UPPAAL, that can verify timed properties without requiring the generation of FSA (even if the results of [15] show that this does not necessary leads to a better performance in the verification).

References

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 1994.

[2] B. Benatallah, F. Casati, J. Ponge, and F. Toumani. On Temporal Abstractions of Web Service Protocols. In *Procs of CAiSE Forum*, 2005.

[3] S. Campos, E. Clarke, W. Marrero, and H. Haraishi. Computing Quantitative Characteristics of Finite-State Real Time Systems. In *Proc. IEEE Real-time systems symposium*, 1994.

[4] Z. Chaochen, C.Hoare, and A. Ravn. A Calculus of Durations. In *IPL*, 1991.

[5] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(4), 2000.

[6] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of web service compositions. In *Proc. of the 18th International Conference on Automated Software Engineering (ASE'03)*, 2003.

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

[8] P. Geguang, Z. Xiangpeng, W. Shuling, and Q. Zongyan. Towards the Semantics and Verification of BPEL4WS. In *Proc. WS-FM'04, ENTCS*, 2004.

[9] R. Kazhamiakin, P. Pandya, and M. Pistore. Timed Modelling and Analysis in Web Service Compositions. In *Proc. ARES'06*, 2006.

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

[11] S. Narayanan and S. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proc. WWW'02*, 2002.

[12] P. Pandya. Specifying and Deciding Qauntified Discrete-time Duration Calculus formulae using DCVALID. In *Proc. Real-Time Tools, RTTOOLS'2001*, 2001.

[13] P. Pandya. Finding extremal models of discrete duration calculus formulae using symbolic search. In *Proc. AVOCS'2004*, 2004.

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

[15] D. Thomas, P. Pandya, and S. Chakraborty. Scheduling clusters in model checking of real-time systems. Technical Report TR-04-16, CFDVS, IIT Bombay, September 2004.