Formal Modeling and Discrete-Time Analysis of BPEL Web Services

Radu Mateescu[1] and Sylvain Rampacek[2]

[1] Inria / Vasy, Centre de Recherche Grenoble – Rhône-Alpes, France
Radu.Mateescu@inria.fr
[2] Le2i, Université de Bourgogne, Dijon, France
Sylvain.Rampacek@u-bourgogne.fr

Abstract. Web services are increasingly used for building enterprise information systems according to the Service Oriented Architecture (Soa) paradigm. We propose in this paper a tool-equipped methodology allowing the formal modeling and analysis of Web services described in the Bpel language. The discrete-time transition systems modeling the behavior of Bpel descriptions are obtained by an exhaustive simulation based on a formalization of Bpel semantics using the Algebra of Timed Processes (Atp). These models are then analyzed by model checking value-based temporal logic properties using the Cadp toolbox. The approach is illustrated with the design of a Web service for Gps navigation.

Keywords: Web services, formal specification, model checking, exhaustive simulation, process algebra.

1 Introduction

Information systems present in companies and organizations are complex software artifacts involving concurrency, communication, and coordination among various applications that exchange data and participate to business processes. *Service Oriented Architecture* (Soa) [15] is a state-of-the-art methodology for developing information systems by structuring them in terms of services, which can be distributed and composed over a network infrastructure to form complex business processes. Web services are a useful basis for implementing business processes, either by wrapping existing software or by creating new functionalities as combinations of simpler ones. Bpel (*Business Process Execution Language*) [14] is a standardized language of wide industrial usage for describing abstract business processes and detailed Web services. It allows to capture both the behavioral aspects (concurrency and communication) and the timing aspects (duration of activities) of Web services.

The Bpel language allows to create Web services either from scratch, or as the composition of existing sub-services, which can be invoked sequentially (one at a time) or concurrently (several ones at the same time). Each Web service described in Bpel can be used as a sub-service by other Web services (described in Bpel or not), thus enabling a hierarchical construction of complex Web services. A Bpel business process is defined by a workflow consisting of various

steps, which correspond internally to algorithmic computations (possibly with time constraints) and externally to message-passing interactions with a client. Business processes are typically built upon existing Web services (although this is not mandatory), each one being specialized for carrying out a particular task. These sub-services are invoked every time a specific information is needed during a step of the workflow; therefore, a business process is not simply the set of sub-services it is built upon, but acts as an orchestrator of these sub-services in order to provide newly added functionalities.

The conjunction of concurrency and timing constraints makes business processes complex and requires a careful design in order to avoid information losses and to obtain a satisfactory quality of service. In this context, formal modeling and analysis techniques available from the domain of concurrent systems allow to improve the quality of the design process and to reduce the development costs by detecting errors as soon as possible during the business process life cycle. These techniques can operate successfully on languages equipped with a formal semantics definition, from which suitable models can be constructed and analyzed automatically.

In this paper, we propose a tool-supported approach for the formal modeling and analysis of business processes and Web services described in Bpel. Our approach consists of the following ingredients: the definition of a formal semantics of Bpel in terms of process algebraic rules, taking into account the discrete-timing aspects [11, 12]; the automated generation of models (state/transition graphs) from the Bpel specifications using an exhaustive simulation based on the formal semantics rules, implemented in the WSMod tool; and the analysis of the resulting models by using standard verification tools for concurrent systems, such as Cadp [9]. We illustrate the application of this approach to the design and discrete-time analysis of a Web service for Gps navigation.

Related work. The modeling and analysis of Web services benefits from a considerable attention in the research community. The Wsat tool proposed in [5, 6] gives to Web service designers the possibility of verifying Ltl properties on Bpel business processes by applying the Spin model checker. Each Bpel process is transformed into a Promela model (via a pattern) and connected to other processes in the description. This work covers only the untimed aspects of Bpel.

Another approach, proposed in [28], uses the Cress (*Chisel Representation Employing Systematic Specification*) notation for specifying the untimed behavior of Web services. Cress descriptions are translated into the formal description technique Lotos [13] and analyzed with dedicated tools, such as Topo, Lola or Cadp. A direct translation from Bpel to Lotos is given in [4], enabling the use of the aforementioned tools for analyzing the untimed behavior of Web services. Bpel was also used as target language for producing executable Web services from Lotos specifications [2, 25]; this allows to combine the advantages of the formal verification using Cadp and of the deployment and execution features of Bpel.

Compared to existing work, our approach differs in the following respects: it is based on a translation of BPEL directly into state/transition graphs, without using an intermediate language such as PROMELA or LOTOS, thus being potentially more efficient; and it handles not only the behavioral, but also the discrete-time aspects of BPEL descriptions.

Paper outline. Section 2 presents our methodology and software platform for modeling and analyzing BPEL descriptions. Section 3 describes the GPS Web service case-study and its analysis using the platform. Finally, Section 4 gives some concluding remarks and directions for future work.

2 Modeling and Analysis Approach

Web services can be seen as complex distributed systems that communicate by message-passing. Therefore, their design methodology can be naturally supported by the formal modeling and analysis techniques stemming from the domain of concurrent systems. To apply these techniques, it is necessary to represent the dynamic behavior of Web services in a formal, non-ambiguous manner.

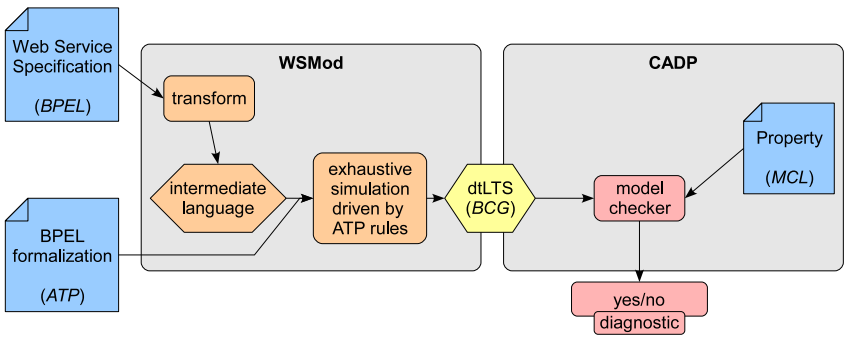


Fig. 1. Platform for Web service modeling and analysis

The approach we propose for the modeling and analysis of Web services described in BPEL is illustrated in Figure 1. Our software platform consists roughly of two parts, described in the sequel: the BPEL descriptions are first translated into discrete-time LTSs using the WSMOD tool, and are subsequently analyzed using the CADP verification toolbox.

2.1 Translation from BPEL to discrete-time LTSs

The behavior of a Web service comprises not only the concurrency and communication between its various constituent activities, but also the delay of response

of the service. These aspects can be modeled using DTLTSs (*discrete-time Labeled Transition Systems*), i.e., state/transition graphs in which every transition is labeled by an action performed by the Web service. The actions are of the following kinds: emissions and receptions of messages, prefixed by '!' and '?', respectively; elapsing of time, represented by the symbol χ (discrete-time tick, also noted *time*); the internal action τ (or *tau*) denoting unobservable activity of the service; and the terminating action $\surd$ (or *done*), which is the last internal action that a service can do.

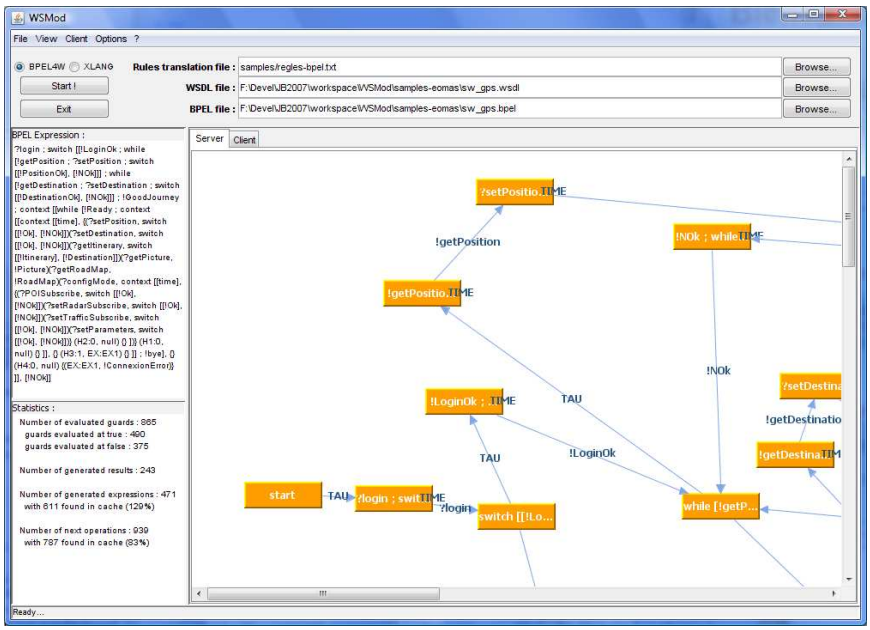


Fig. 2. Screenshot of the WSMOD tool

The global behavior of the Web service (and therefore, the actions it can perform) is obtained by an exhaustive simulation of the BPEL description, performed by the WSMOD tool (see a screenshot in Figure 2), which is able to handle both discrete [11] and continuous [12] time representations. WSMOD takes two different inputs (see Figure 1):

– A Web service description in BPEL [14], a standardized language allowing to specify the behavior of business processes. BPEL supports two different types of business processes: *executable* processes specify the behavior of business processes in full detail, such that they can be executed by an orchestration engine; and *abstract* business protocols specify the public message exchanges between the service and a client (i.e., excluding the message exchanges which take place internally, e.g., during invocations of sub-services).

– A formal representation of the BPEL semantics, based on the Algebra of Timed Processes (ATP) [24], which specifies using operational rules how the model of the business process behavior is generated. Depending on the time representation chosen, the resulting model is either a DTLTS, or a timed automaton (TA) [1]. An excerpt of the ATP rules formalizing the BPEL semantics in discrete-time is shown in Table 1. For example, the process "time" can only elapse time (represented by the χ action), and the process "receive" or "reply" can send or receive a message (first rule) or elapse time too (second rule).

To generate the model representing the behavior of the input BPEL description, WSMOD performs an exhaustive simulation guided by the operational ATP rules. The tool is also able to synthesize automatically the model of an adapted client interacting with the Web service, whose behavior complies with that of the service as regards emission and reception of messages, time elapsing, etc. In this study, we focus only on the Web service model generation feature of WSMOD.

2.2 Analysis of discrete-time LTSs

Once the DTLTS model of the BPEL specification under design has been obtained, it can be analyzed by using standard tool environments available for concurrent systems. For our purpose, we use the CADP (*Construction and Analysis of Distributed Processes*) toolbox [9] dedicated to the formal specification and verification of concurrent asynchronous systems. CADP accepts as input specifications written in process algebraic languages, such as LOTOS [13], FSP [18, 26] or CHP [19, 27], as well as networks of communicating automata given in the EXP language [16]. These formal specifications are translated by specialized compilers into labeled transition systems (LTSs), i.e., state spaces modeling exhaustively the dynamic behavior of the specified systems. LTSs are the formal model underlying the analysis functionalities offered by CADP, which aim at assisting the user throughout the whole design process: code generation and rapid prototyping, random execution, interactive and guided simulation, model checking and equivalence checking, test case generation, and performance evaluation.

An LTS can be represented within CADP in two complementary ways: either *explicitly*, by its list of states and transitions encoded as a file in the BCG (*Binary Coded Graphs*) format equipped with specialized compression algorithms, or *implicitly*, by its successor function given as a C program complying to the interface defined by the OPEN/CÆSAR [7] environment for graph manipulation. The explicit representation is suitable for *global* verification algorithms, which explore transitions forward and backward, whereas the implicit representation is suitable for *local* (or *on-the-fly*) verification algorithms, which explore transitions forward, thus enabling an incremental construction of the LTS during verification. To deal with large systems, CADP provides several advanced analysis techniques: on-the-fly verification, partial order reductions, compositional verification, and massively parallel verification using clusters of machines.

Table 1. An extract of the process algebra formalizing BPEL, in discrete-time

BPEL	**ATP**
empty	$\text{empty} \xrightarrow{\surd} 0$
time	$\text{time} \xrightarrow{\chi} \text{time}$
throw	$\forall\, e \in E_X,\ \text{throw}[e] \xrightarrow{e} 0$ with E_X set of exceptions that can be thrown.
receive / reply	$*o[m] \xrightarrow{*m}$ empty with $* \in \{?, !\}$ $*o[m] \xrightarrow{\chi} *o[m]$
sequence (;)	$\forall\, a \neq \surd,\ \dfrac{P \xrightarrow{a} P'}{P\ ;\ Q \xrightarrow{a} P'\ ;\ Q}$ $\forall\, a,\ \dfrac{P \xrightarrow{\surd} P' \quad \wedge \quad Q \xrightarrow{a} Q'}{P\ ;\ Q \xrightarrow{a} Q'}$
switch	$\text{switch}[\{P_i\}_{i \in I}] - \forall\, i \in I,\ \text{switch}[\{P_i \mid i \in I\}] \xrightarrow{\tau} P_i$
while	$\text{while}[P] \xrightarrow{\tau} P\ ;\ \text{while}[P]$ $\text{while}[P] \xrightarrow{\tau} \text{empty}$
scope	Let $M_I = \{m_i \mid i \in I\}$ a set of messages and let $E_J = \{e_j \mid j \in J\}$ a set of exceptions. $\text{scope}(P, E)$ with $E = [\{(m_i, P_i) \mid i \in I\}, (d, Q), \{(e_j, R_j) \mid j \in J\}]$ $\dfrac{P \xrightarrow{\surd}}{\text{scope}(P,E) \xrightarrow{\surd} 0}$ $\forall\, a \notin \{\chi, \surd\} \cup E_X \cup M_I \quad \dfrac{P \xrightarrow{a} P'}{\text{scope}(P,E) \xrightarrow{a} \text{scope}(P', E)}$ $d > 1,\ \dfrac{P \xrightarrow{\chi} P' \ \text{ and } \ \forall\, a \in E_X \cup \{\tau, \surd\},\ \neg(P \xrightarrow{a})}{\text{scope}(P, E^d) \xrightarrow{\chi} \text{scope}(P, E^{d-1})}$ $\dfrac{P \xrightarrow{\chi} P' \ \text{ and } \ \forall\, a \in E_X \cup \{\tau, \surd\},\ \neg(P \xrightarrow{a})}{\text{scope}(P, E^1) \xrightarrow{\chi} Q}$ $\forall\, i \in I,\ \dfrac{\forall\, a \in E_X \cup \{\tau, \surd\},\ \neg(P \xrightarrow{a})}{\text{scope}(P,E) \xrightarrow{?m_i} P_i}$ $\forall\, j \in E_J,\ \dfrac{P \xrightarrow{e_j}}{\text{scope}(P,E) \xrightarrow{\tau} R_j}$ $\forall\, e \notin E_J,\ \dfrac{P \xrightarrow{e}}{\text{scope}(P,E) \xrightarrow{e} 0}$
pick	$\text{pick}[E] = \text{scope}(\text{time}, E)$ with $E = [\{(m_i, P_i) \mid i \in I\}, (d, Q), \{(e_j, R_j) \mid j \in J\}$

CADP contains currently over 40 tools and libraries for LTS manipulation, which can be invoked either in interactive mode via the EUCALYPTUS graphical interface, or in batch mode via the SVL [8] scripting language dedicated to the description of complex verification scenarios. The toolbox was used for the validation of more than 100 industrial case-studies[3].

Since we focus on model checking discrete-time properties on DTLTSs, we could apply in principle existing tools operating on LTSs, such as the EVALUATOR 3.5 [20] on-the-fly model checker of CADP, which takes as input temporal formulas expressed in regular alternation-free μ-calculus. However, the evaluation of discrete-time properties requires the counting of *time* actions in the DTLTS; this can be encoded in μ-calculus using fixed point operators (one operator for each counter value), but may lead to prohibitively large temporal formulas, as noticed in the framework of temporal CCS [23]. Discrete-time properties can be naturally formulated using data-handling extensions of the modal μ-calculus, such as the MCL language [21] underlying the EVALUATOR 4.0 tool recently integrated into CADP. We will illustrate the usage of MCL in Section 3.3.

3 Case Study: A Web Service for GPS Navigation

We illustrate in this section the application of our approach to the modeling and analysis of a Web service dedicated to GPS navigation. Given the relative complexity of this Web service, we do not detail here its textual BPEL and WSDL descriptions, but present its workflow graphically using the BPMN [10] notation.

3.1 System description

The purpose of the GPS Web service is to compute itineraries from a position to a destination fixed by a user (client of the service). In addition to the requested itinerary, the user can also obtain: pictures of the travel (taken from the air), the global map of the itinerary, and various kinds of information (about traffic, radar stations, point of interest (POI), etc). At last, the user can configure the subscription to the various kinds of information, as well as some parameters of the travel (e.g., to take motorway or not, to deviate toward a POI, etc.). The relationships between these functionalities are represented in Figure 3.

The behavior of the GPS Web service consists of two main phases, described in the sequel: the initialization phase (login, setting of the initial position and destination) and the main loop phase (management of the itinerary, modification of the parameters, etc.).

Initialization phase The initialization phase comprises three activities: login, position and destination.

[3] See the online catalog at `http://www.inrialpes.fr/vasy/cadp/case-studies`

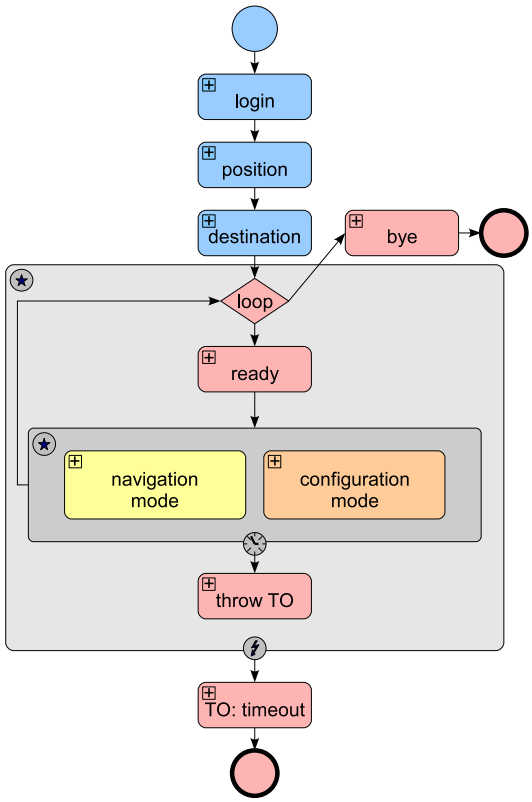
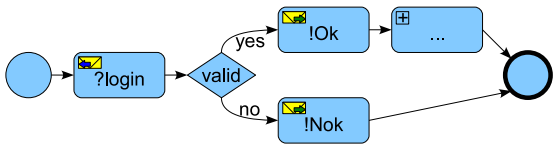
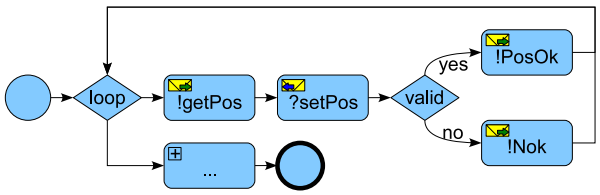


Fig. 3. Functionality workflow of the GPS Web service

Login activity. The access to the Web service is restricted to authenticated users only. To identify itself, the user must send a couple login/password, to which the service responds by a message "Ok" or "NOk" depending whether the couple is valid or not.



Position activity. After authentication, in order to use the main functionalities of the Web service, the user must indicate where the start location of the travel is. This is done by sending a message with information about the street, city, and country where the navigation session must be started; the message must be resent until the start location is accepted by the service (message "Ok").



Destination activity. Finally, before the service may attempt to calculate an itinerary, the user must enter a destination. This is similar to the position activity above: the user must retry until the end location is accepted by the service.

Main loop phase After the initialization phase, the service can compute an itinerary, send information about traffic, POI, etc. To maintain the connection with the user, the service requires that the time elapsed between certain consecutive user actions does not exceed a given timeout value (a kind of "ping alive"). In Section 3.2 we will consider for analysis configurations of the system with a timeout value ranging from 1 to 60 seconds.

From the Web service point of view (see Figure 3), this timeout is managed by a "scope" process: when the timeout is reached, this process generates an exception that will be caught by another process. The main activity of this "scope" process is a "pick" process. This kind of choice enables the user to select a desired action; if we used the "switch" BPEL construct instead, the choice would be made by the Web service and not by the user. Furthermore, the "scope" is encapsulated into a "while", enabling the user to do more than one operation during the session (notice that the first action carried out by the service when entering the "while" is the emission of a *ready* message to the user). Finally, the "while" is the main activity of a second "scope", that catches the first exception thrown when the timeout is reached.

The activities executed by the main loop are partitioned in two modes, described in the sequel: the navigation mode (obtaining the itinerary, modifying the current position or destination, getting a picture or a roadmap), and the configuration mode (subscribing to a POI, getting information on radars or traffic, setting of parameters).

Navigation mode. In navigation mode, the user can change the current position and the destination (using the same procedure as for the initialization phase). Next, the user can ask for the itinerary, a picture, the roadmap, or enter in configuration mode. There are two types of answer for itinerary requests: either a complete itinerary leading from the current position to the destination, with various information (about street, radar, POI, etc.) depending on the user subscriptions, or simply a *destination* message indicating that the current position is (near) to the destination. The requests for picture and roadmap allow the user to obtain an air-picture of the area (in PNG format) or a veritable roadmap (in SVG format).

Configuration mode. In configuration mode, the user can subscribe or cancel his subscription to information about POI, radar or traffic. This information is added to the itinerary if necessary. Additionally, the user can set various parameters, such as the kind of the itinerary (on motorway or not), etc.

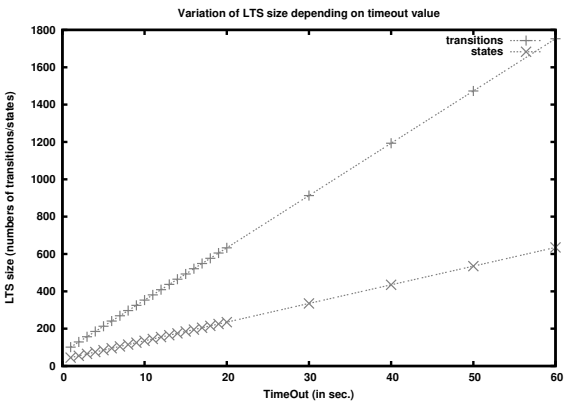
3.2 Discrete-time LTS synthesis

Starting from the BPEL description of the GPS Web service, we apply the WSMOD tool in order to obtain a DTLTS on which the verification tools of CADP will operate. We show below the DTLTS model obtained for a timeout of 1 second, then we study its variation in size as the timeout value increases, and finally we discuss the behavior of the Web service w.r.t. the ambiguity detection feature implemented in WSMOD.

Discrete-timed labeled transition system. DTLTS models represent the observable behavior of Web services. The actions labeling the DTLTS transitions denote the messages exchanged (emissions and receptions are prefixed by '!' and '?', respectively), the elapse of a discrete-time unit χ (or *time*), the internal action τ (or *tau*), and the termination action $\surd$ (or *done*). The global behavior of the Web service is obtained by an exhaustive simulation of the BPEL description driven by the ATP rules given in Table 1. The DTLTS obtained in this manner for the GPS Web service with a timeout value of 1 second is shown in Figure 4.

Variation of DTLTS size with the timeout value. The size of the DTLTS (number of states and transitions) depends on several aspects of the BPEL description: the number of BPEL processes, their complexity and nesting, the amount of communications, and the values of the timeouts. For the sake of readability, we have shown in Figure 4 the DTLTSs for a timeout of 1 second (corresponding to one χ in discrete-time), but we carried out verification also for larger values of the timeout.

The figure on the right shows the variation of the DTLTS size for timeout values ranging from 1 to 60. We observe a linear increase of both the number of states and transitions; this is a consequence of the fact that the BPEL description contains a single timeout (according to the ATP rules). In the presence of multiple, overlapped timeouts, the size of the DTLTS may increase much more quickly.



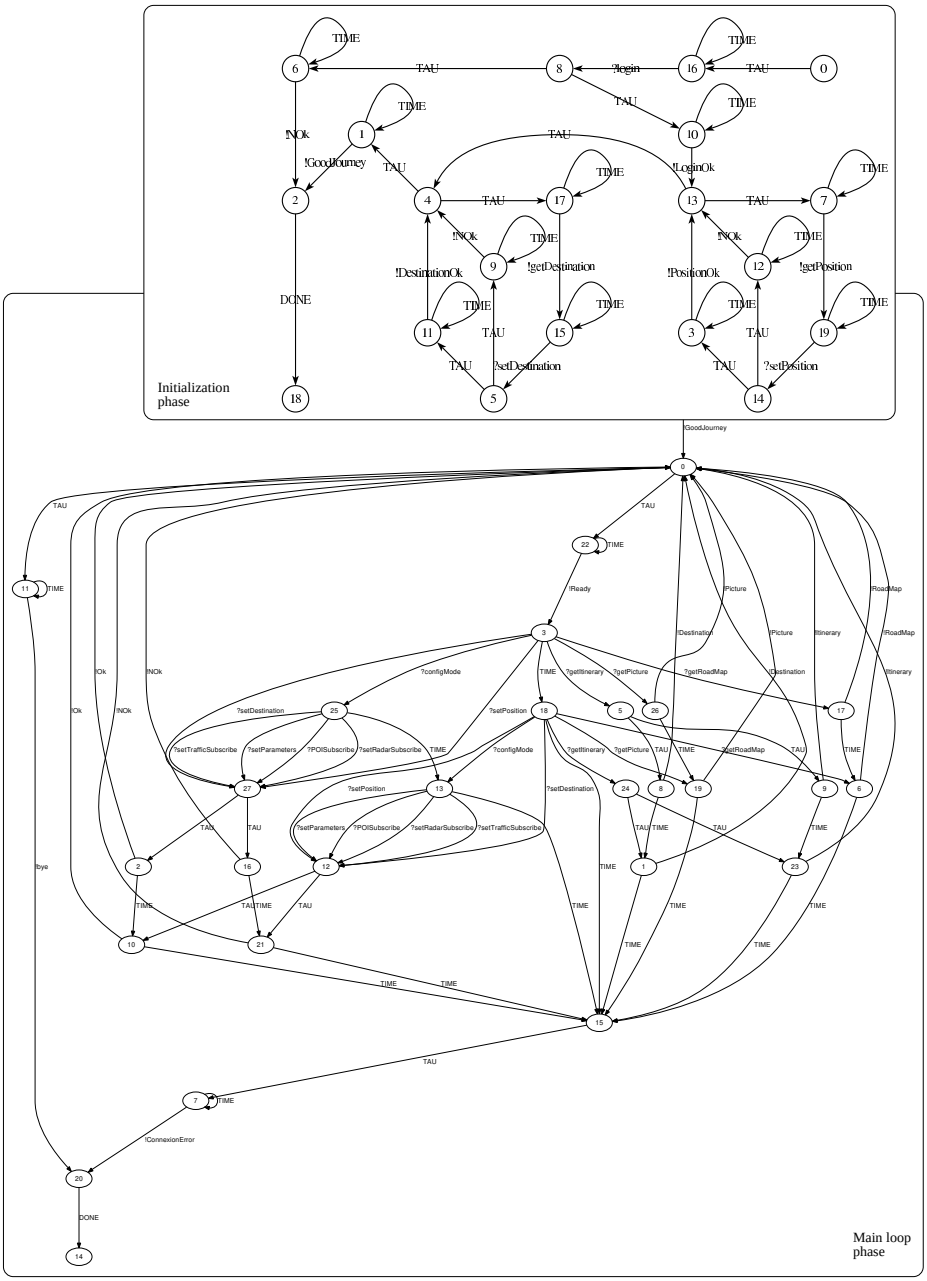


Fig. 4. DTLTS model of the GPS Web service, with zoom on the initialization phase. The action *!GoodJourney* makes the link between the initialization phase and the main loop phase.

Non ambiguous Web service. In this study, we focus on the verification of the Web service behavior. However, the WSMod tool can also synthesize automatically a dtLts modeling the behavior of an adapted client interacting with the Web service, provided that the model of the service respects certain properties (concerning non ambiguity in message exchanges, time elapsed, etc.) detailed in [11]. Here, the Gps Web service is identified as *non ambiguous* by WSMod, meaning that the tool can synthesize an adapted client that can know, on each message exchange, the exact choice made on the service side, and therefore the client and the service can evolve without any deadlocks or mismatches.

3.3 Verification of discrete-time properties

We analyze below the behavior of the Gps Web service (considering a time-out of 50 seconds) by means of discrete-time model checking using the Evaluator 4.0 [21] tool of Cadp. Table 2 illustrates the formulation in Mcl of several safety and liveness properties, of both untimed and timed nature. The colored parts of the formulas indicate discrete-time properties, which involve the counting of *time* actions. All properties were successfully verified on the corresponding dtLts of the system, which has 535 states and 1473 transitions.

Table 2. Safety and liveness properties of the Gps Web service (timeout of 50 sec.)

Prop.	Mcl formula
S_1	$[\ (\neg !LoginOk)^*.?setPosition \vee ?setDestination\ ]$ false
S_2	$[\ (\mathsf{true}^*.$ $((!getPosition.(\neg ?setPosition)^*) \mid (!getDestination.(\neg ?setDestination)^*)).$ $!GoodJourney)\ ]$ false
S_3	$[\ \mathsf{true}^*.?getItinerary.(\neg(!Itinerary \vee !Destination))^*.$ $(?getPicture \vee ?getRoadMap \vee ?configMode \vee$ $?setPosition \vee ?setDestination \vee ?getItinerary)\ ]$ false
S_4	$[\ \mathsf{true}^*.?getItinerary.(\neg(!Itinerary \vee !Destination))^*.$ $(time.(\neg(!Itinerary \vee !Destination))^*)\{51\}\ .$ $(!Itinerary \vee !Destination)\ ]$ false
L_1	$[\ \mathsf{true}^*.!LoginOk\ ]\ \mathsf{AF}\ \langle !getPosition \vee !getDestination \vee !GoodJourney\rangle\ \mathsf{true}$
L_2	$[\ \mathsf{true}^*.!GoodJourney.(\tau \vee time)^*\ ]\ \langle(\tau \vee time)^*.\ !Ready \vee !bye\rangle\ \mathsf{true})$
L_3	$[\ \mathsf{true}^*.!Ready.\ time\{\ ...\ 50\ \}\ ]$ $\langle \mathsf{true}^*.!Picture \vee !RoadMap \vee !Itinerary \vee !Destination\rangle\ \mathsf{true}$
L_4	$[\ \mathsf{true}^*.!Ready.$ $((\neg(!Itinerary \vee !Destination \vee !Picture \vee !RoadMap))^*.time)\{51\}\ ]$ $\mathsf{AF}\ \langle !ConnectionError\rangle\ \mathsf{true}$

Safety properties: they specify informally that "something bad never happens" during the execution of the system. In the MCL language, these properties can be expressed in a concise manner by identifying the undesirable execution sequences, characterizing them using extended regular expressions, and forbidding their existence in the DTLTS model using necessity modalities.

Properties S_1 and S_2 concern the ordering of actions during the initialization phase: S_1 specifies that the user cannot set the position or the destination before logging in successfully, and S_2 states that after requesting the position or the destination, the Web service cannot begin the main loop before receiving an appropriate answer from the user. Properties S_3 and S_4 deal with the main loop phase: S_3 forbids the user to make another request before the current one (here, an itinerary demand) has been handled by the service, and S_4 states that a demand cannot be fulfilled anymore by the service after the timeout has expired. The $R\{...\,n\}$ and $R\{n\}$ extended regular operators denote the repetition of a regular expression R at most n times and exactly n times, respectively.

Liveness properties: they specify informally that "something good eventually happens" during the execution of the system. In MCL, these properties contain diamond modalities and minimal fixed point operators for encoding the existence of certain desirable execution sequences (*potentiality*) or trees (*inevitability*) in the DTLTS.

Properties L_1 and L_2 concern the initialization phase: L_1 specifies that after the user has logged in, the Web service will eventually ask for the position, the destination, or end the initialization, and L_2 states that after the initialization was finished the service will end up in the main loop or decide to terminate the session. Properties L_3 and L_4 deal with the main loop phase: L_3 indicates that as long as the timeout has not expired, the service can still prompt for a user request, and L_4 states that an expiration of the timeout eventually interrupts the connection. The $\mathsf{AF}\ p$ operator of CTL [3] expressing the inevitable reachability of a state p is defined in μ-calculus as $\mu X.p \vee (\langle \mathsf{true} \rangle\, \mathsf{true} \wedge [\mathsf{true}]X)$.

4 Conclusion and Future Work

The design of complex business processes according to the SOA approach requires to carefully take into account the presence of concurrency, communication, and timing constraints induced by the interaction of Web services. To facilitate the design process, we propose here a tool-equipped methodology for modeling and analyzing Web services described in BPEL. We focus on the behavioral and discrete-time aspects of Web services, and rely upon the model-based verification technologies stemming from the concurrency domain. The state/transition models of BPEL Web services are produced automatically by the WSMOD tool, which implements an exhaustive simulation algorithm based on a formalization of BPEL semantics by means of process algebraic rules. The tool is able to handle both discrete and continuous time constraints; for the moment we handle only discrete-time models, which can be analyzed using the EVALUATOR 4.0 model

checker [21] of the CADP toolbox [9]. Discrete-time safety and liveness properties can be concisely expressed using the data-handling facilities of the MCL language accepted as input by EVALUATOR 4.0, and particularly the extended regular expressions over transition sequences, which allow to count tick actions occurring in the model. We illustrated the verification of discrete-time properties on the example of a GPS Web service; however, most of them can be easily adapted for other business processes described in BPEL. Our methodology enables the Web service designers to carry out formal analysis on complex Web services before publishing them, and thus to improve the quality of the design process.

We plan to continue our work along several directions. Firstly, we can improve the connection between WSMOD and CADP by producing implicit DTLTSs according to the interface defined by OPEN/CÆSAR [7]. This would enable on-the-fly verification, which allows to detect errors in large systems without constructing the complete DTLTS model beforehand but exploring it in a demand-driven way. Secondly, using discrete-time models allows to directly reuse the tools available for data-based temporal logics, such as EVALUATOR 4.0; however, this may lead to state explosion in the presence of numerous timeouts. An alternative solution would be to use continuous time models; this can be achieved by connecting the time automata produced by WSMOD with the UPPAAL [17] tool dedicated to the verification of continuous time models. Finally, we will extend the methodology to handle compositions of multiple Web services, following our previous work on automated client synthesis [22], but focusing on the verification of composition. For this purpose, the compositional verification techniques available in tools such as EXP.OPEN [16] will be certainly useful.

References

1. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. A. Chirichiello and G. Salaün. Encoding abstract descriptions into executable web services: Towards a formal development. *Proc. of WI '05*, pages 457–463, IEEE Computer Society.
3. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.
4. A. Ferrara. Web services: a process algebra approach. In *ICSOC*, pages 242–251, 2004.
5. X. Fu, T. Bultan, and J. Su. Analysis of interacting BPEL web services. In *Proc. of the 13th International World Wide Web Conference (WWW'04)*, USA, 2004. ACM Press.
6. X. Fu, T. Bultan, and J. Su. WSAT: A tool for formal analysis of web services. In *Proc. of the 16th International Conference on Computer Aided Verification (CAV'04)*, 2004.
7. H. Garavel. OPEN/CÆSAR: An open software architecture for verification, simulation, and testing. *Proc. of TACAS'98*, LNCS vol. 1384, pp. 68–84, March 1998. Springer Verlag. Full version available as INRIA Research Report RR-3352.
8. H. Garavel and F. Lang. SVL: a scripting language for compositional verification. *Proc. of FORTE'2001*, pp. 377–392. IFIP, Kluwer Academic Publishers, August 2001. Full version available as INRIA Research Report RR-4223.

9. H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2006: A toolbox for the construction and analysis of distributed processes. *Proc. of CAV'2007*, LNCS vol. 4590, pp. 158–163. Springer Verlag, July 2007.
10. Object Management Group. Business process modeling notation (BPMN) specification, may 2006.
11. S. Haddad, T. Melliti, P. Moreaux, and S. Rampacek. Modelling web services interoperability. In *Proc. of the 6th Int. Conf. on Enterprise Information Systems (ICEIS04)*, Porto, Portugal, April 14–17 2004.
12. S. Haddad, P. Moreaux, and S. Rampacek. A formal semantics and a client synthesis for a BPEL service. In *Lecture Notes in Business Information Processing, ICEIS06 Revised Selected Paper*, volume 3. Springer, 2008.
13. ISO/IEC. LOTOS — a formal description technique based on the temporal ordering of observational behaviour. International Standard 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, September 1989.
14. D. Jordan and J. Evdemon. Web Services Business Process Execution Language Version 2.0 - Oasis Standard, 11 april 2007.
15. N. Josuttis. *Soa in Practice – The Art of Distributed System Design*. O'Reilly Media, City, 2007.
16. F. Lang. EXP.OPEN 2.0: A flexible tool integrating partial order, compositional, and on-the-fly verification methods. *Proc. of IFM'2005*, LNCS vol. 3771. Springer Verlag, 2005.
17. K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.
18. J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. Wiley, 1999.
19. A. J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1(4):226–234, 1986.
20. R. Mateescu and M. Sighireanu. Efficient on-the-fly model-checking for regular alternation-free mu-calculus. *Science of Computer Programming*, 46(3):255–281, March 2003.
21. R. Mateescu and D. Thivolle. A model checking language for concurrent value-passing systems. *Proc. of FM'2008*, LNCS vol. 5014. Springer Verlag, 2008.
22. T. Melliti, C. Boutrous-Saab, and S. Rampacek. Verifying correctness of web services choreography. *Proc. of ECOWS06*, Zurich, Switzerland, December 4–6 2006. IEEE Computer Society Press.
23. M. J. Morley. Safety-level communication in railway interlockings. *Science of Computer Programming*, 29(1-2):147–170, 1997.
24. X. Nicollin and J. Sifakis. The algebra of timed processes ATP: Theory and application, 1994.
25. G. Salaün, A. Ferrara, and A. Chirichiello. Negotiation among web services using LOTOS/CADP. *Proc. of ECOWS04*, LNCS vol. 3250, pages 198–212. Springer, 2004.
26. G. Salaün, J. Kramer, F. Lang, and J. Magee. Translating FSP into LOTOS and networks of automata. *Proc. of IFM'2007*, LNCS vol. 4591, pp. 558–578. Springer Verlag, July 2007.
27. G. Salaün and W. Serwe. Translating hardware process algebras into standard process algebras — illustration with CHP and LOTOS. *Proc. of IFM'2005*, LNCS vol. 3371, pp. 287–306. Springer Verlag, 2005.
28. K. J. Turner. Representing and analysing composed web services using CRESS. *J. Netw. Comput. Appl.*, 30(2):541–562, 2007.