Bounded Liveness in Semantic Web Services
Amel Boumaza, Ramdane Maamri
Lire Laboratory, Computer Sciences Department, University Mentouri Constantine, Algeria

Abstract: Timing constraints verification for service flow has become indispensable since many services for e-business take into account temporal context. Currently, most researches disregard temporal factors. In this paper, we focus on bounded liveness property, one of most interesting timed properties. Specifically, we propose a modelling procedure that preserves the control flow of OWL-S Process Models. By using time ontology based on DAML-Time, OWL-S is extended with time constraints, which expresses time information. The Timed OWL-S is transformed to Timed Automata formal model, then we apply the automatic tool Uppaal for the verification of the bounded liveness property.

Keywords: Semantic Web Services, OWL-S, Time Ontology, Model Checking, Timed Automata, Bounded Liveness.

1. Introduction

Web Services extend the Web from a distributed source of information to a distributed source of services. The Semantic Web has added machine-interpretable information to Web content in order to provide intelligent access to heterogeneous and distributed information. In a similar way, Semantic Web concepts are used to define intelligent Web services i.e., services supporting automatic discovery, composition, invocation and interoperation. This joint application of Semantic Web concepts and Web Services in order to realize intelligent Web Services is usually referred as Semantic Web Services (Lara et al., 2003).

At present, Web Services can be described by a high-level modeling language such as WSFL (Web Service Flow Language) and BPEL (Business Process Execution Language). However, they emphasize the execution process of service and can not express service semantic exactly.

In order to cope with this need, the emerging convergence of Web Services with Semantic Web, conventionally Semantic Web Services, aim to describe and implement Web Services so as to make them more accessible to automated agents. Ontology languages such as DAML+OIL [7] and OWL [4] provide basic vocabularies for describing concepts and relationships. Hence, agents can perform reasoning without human guidance.

In Semantic Web Services, the common need for temporal information is satisfying by the use of the temporal ontologies to allow using of temporal concepts. However, complexity comes usually with the growing expressiveness; it becomes a challenging area to ensure correctness. Hence, the verification of Web Service flow becomes more and more important.

A Web process defines a set of activities and the specific order they are to be executed to achieve a common goal. When modeling and analyzing systems based on Web processes, time related information is often an important factor. Time consistency verification of Web processes can be used for guaranteeing that the timing constraints are met.

For this reason, we propose to use the software engineering techniques and tools, i.e., Timed Automata, to complement ontologies for specifying and checking temporal properties.

2. Related Work

Verification and validation of Web services is an emerging area. Related work mainly includes: Shin proposed the automation-based model checker SPIN to verify the service flows described by WSFL [12]. Howard et al. applied FSM (Finite State Machine) notation to model and verify service compositions that are specified using BPEL4WS [5], while X. Yi and K. J. Kochut introduced a CP-nets model, an extended Petri-Net model, and argue that Petri-Net is more expressive than FSM for specifying conversational protocols [14]. Particularly, Srinivasa and Sheila adopted DAML-S ontology to describe service semantics, translated DAML-S semantic specification into a Petri-Net specification, and tried to test Web Service by simulating its execution under different input conditions [13]. Other researchers also conducted related research in different way. However, their verification methods still lack consideration for time factor.

Some works have dealt with qualitative properties. In [6], authors use a WS-CDL (Web Services Choreography Description Language) description of composite Web services, and define an operational semantics for a translation of a subset of WS-CDL.
into a network of Timed Automata. Uppaal tool is used for the validation and verification of the generated timed automata. In [6], authors deal with the compatibility problem. The proposed framework allow applying model checker Uppaal to asynchronous Web services composition analysis by using a set of CTL formulas that characterize the different choreography compatibility classes defined and verifying deadlock free property.

Particularly, Rujuan and al. [10] developed a method for verifying temporal consistency in Web Service flow. Time ontology is added on OWL-S specification, and then the annotated OWL-S is transformed into formal model TCPN. By computing time constraints of Web Service with different composition, and analysing relevant properties of TCPN, temporal consistency of Web Service flow can be verified.

It can be seen that there is few work on timing constraint satisfiability based on formal methods and witch can combine semantic web approach. On another hand and to our knowledge, there is no research conducted on bounded liveness property verification.

In this paper, we present a method for verifying the bounded liveness property on Timed OWL-S. Time ontology is combined with OWL-S specification and then the timed OWL-S is transformed into timed automata model. Then, we use Uppaal model checker for verification.

The rest of the paper is structured as follows. Section 3 briefly introduces OWL-S Ontology. Section 4 presents the concept of Timing Constraints specification. Section 5 introduces basic timed model checking technique. Section 6 depicts the OWL-S bounded liveness verification approach. Section 7 presents the multimodel transport case study. Section 8 concludes the paper and discusses future work.

3. OWL-S Overview

OWL-S is a high level ontology-based language for describing web service properties. In this language, each web service is specified in three XML-based parts:

- Service Profile, which describes what the service does?
- Service Model, which describes how does the service work (behave);
- Service Grounding, which provides details on how to invoke a service via messages [11].

OWL-S allows the description of the external behaviour of a Web service by using a semantic model, in which each involved atomic process is described semantically in terms of inputs, preconditions, outputs, and effects (IPOEs).

Composite processes are used to describe collections of processes (either atomic or composite) organized on the basis of following control flow structure: (1) Sequence: all components are performed sequentially, (2) If-then-else: the selection based on some clearly defined conditions, (3) Choice: the selection depends on some information, (4) Repeat-while: the component is performed repeatedly while certain conditions are satisfied, (5) Repeat-until: the component is performed repeatedly until some conditions hold, (6) Anyorder: the components are performed in an order decided at runtime, (7) Split: specifies that the components are activated and performed in parallel, (8) Split-Join: specifies that a component is not activated until the parallel execution of the preceding components terminates.

4. Timing Constraint Specification

To provide support for describing temporal properties for OWL-S (formerly DAML-S), we use an "entry" sub-ontology of time [8] which is much simpler than the full ontology of DAML-Time [Hobbs 2002] which provides the basic temporal concepts and relations that most simple applications would need, i.e., a vocabulary for expressing facts about topological relations.

The most basic temporal concepts in the sub-ontology are Instant, Interval, Instant Event, and Interval Event. Instants are, intuitively, point-like in that they have no interior points, and intervals are, intuitively, things with extent. Instant events are events that are instantaneous, such as the occurrence of a car accident or the arrival of a package, and interval events are events that span some time interval, for example, a meeting from 2pm to 3pm.

Besides these four basic temporal concepts, there are five other more general temporal concepts/classes: Temporal Thing, Temporal Entity, Instant Thing, Interval Thing, and Event. The subclass hierarchy of these temporal concepts/classes is shown in the Figure 2. For example, Instant Thing has two subclasses: Instant and Instant Event.

By adding timing constraints to OWL_S, the time information related to the service can be defined easily. The entry sub-ontology provides quick access to the essential vocabulary in OWL for the

---

1DAML-Time
Homepage: http://www.cs.rochester.edu/~ferguson/daml/
The basic temporal concepts and relations. It covers topological relations among instants and intervals and instant-like and interval-like events such as "before" and "overlaps". It includes measures for durations so that we can say a meeting will last 1 hour and 30 minutes, and it also includes clock and calendar terms so that we can say a meeting starts at 3:00pm PST on Monday, October 20, 2012.

Figure 2. Subclass hierarchy of temporal concepts

The essential idea behind model checking is shown for checking safety properties distinguished, safety and liveness. In practical formalisms exist. Typically two types of approaches for verifying temporal specifications of model checking tool accepts system requirements or design (called models) and a property (called specification) that the final system is expected to satisfy. The tool then outputs yes if the given model satisfies given specifications and generates a counter example otherwise.

5. Timed Model Checking

Model Checking [2] is one of the most successful approaches for verifying temporal specifications of hardware and software systems. System properties are specified in temporal logic for which various formalisms exist. Typically two types of properties are distinguished, safety and liveness. In practical applications, safety properties are prevalent. Therefore very efficient algorithms and tools have been devised for checking safety properties.

The essential idea behind model checking is shown in Figure 3. A model-checking tool accepts system requirements or design (called models) and a property (called specification) that the final system is expected to satisfy. The tool then outputs yes if the given model satisfies given specifications and generates a counter example otherwise.

Figure 3. Principles of model checking

In this paper, we choose Timed Automata as underlying model, and TCTL [1b] logic to specify the property to verify.

5.1. Timed Automata

In this section we reply Timed Automata, which were introduced by Alur and Dill [1a]. Timed Automata are extensions of finite state automata with constraints on timing behaviour. The underlying finite state automata are augmented with a set of real time variables.

Definition A Timed Automaton is a six-element tuple

\[ \mathcal{A} = (A, L, l_0, E, X, I) \], where

- \( A \) is a finite set of actions,
- \( L \) is a finite set of locations,
- \( l_0 \in L \) an initial location,
- \( X \) is an initial location,
- \( E \subseteq L \times A \times C_X \times 2^X \times L \) a transition relation,
- \( I : L \rightarrow C_X \) is a (location) invariant.

Each element \( e \) of \( E \) is denoted by \( l \xrightarrow{a,cc,X} l' \) which represents a transition from the location \( l \) to the location \( l' \), executing the action \( a \), with the set \( X \subseteq C_X \) of clocks to be reset, and with the clock constraint \( cc \) defining the enabling condition for \( e \). The function \( I \) assigns to each location \( l \in L \) a clock constraint defining the conditions under which \( A \) can stay in \( l \).

Given a transition \( e = l \xrightarrow{a,cc,X} l' \), we write source \( (e) \), target \( (e) \), action \( (e) \), guard \( (e) \) and reset \( (e) \) for \( l, l', a, cc \) and \( X \), respectively. The clocks of a Timed Automaton allow expressing the timing properties. An enabling condition constrains the execution of a transition without forcing it to be taken. An invariant condition permits an automaton to stay at the location \( l \), only as long as the clock constraint \( I(l) \) is satisfied.

5.2. Bounded Liveness

A liveness property is a property stating that "something good will eventually happen".

However, for real-time systems general liveness properties are often not sufficiently expressive to
ensure correctness: the fact that a particular property is
guaranteed to hold eventually is inadequate in case hard
real-time deadlines must be observed. What is really
needed is to establish that the property in question will
hold within a certain upper time-limit. Thus, a bounded
liveness property is a liveness property that comes with
a maximal delay within which the ”good thing” must
occur.

Several versions are available. We consider the more
efficient one. Based on the method proposed in [9] in
which time-bounded leads-to properties are reduced to
simple safety properties, first the model under
investigation is extended with a Boolean variable \( b \) and
an additional clock \( z \). The boolean variable \( b \) must be
initialized to \( false \). Whenever \( \varphi \) starts to hold, \( b \) is set to
\( true \) and the clock \( z \) is reset. When \( \psi \) commences to
hold \( b \) is set to \( false \). Thus the truth-value of \( b \) indicates
whether there is an obligation of \( \psi \) to hold in the future
and \( z \) measures the accumulated time since this
unfulfilled obligation started. The time-bounded leads-
to property \( \varphi \models_{st} \psi, t \in [2] \) is simply obtained by
verifying the safety property \( \forall \Box (b \Rightarrow z \leq t) \).

\[ \exists \Box (b \Rightarrow 2 \leq z \leq 3) \]

Figure 4. Example of TCTL formulae.

Example Intuitively, the formulae \( \varphi \models_{[2,3]} \psi \)
express that for all the runs, always when \( \varphi \) holds, \( \psi \) holds in 2
to 3 units of time. In Figure 4, the formulae
\( \exists \Box (b \Rightarrow 2 \leq z \leq 3) \) holds for all states of one run. If
this is true for all other runs then the formulae
\( \varphi \models_{[2,3]} \psi \) holds too.

6. Generic Timed OWL-S Verification Approach

The paper describes how to model Web Services
and determine whether the specification satisfies
the bounded liveness property. In the following we
present different steps of the proposed approach:

- **Step 1**: we model a Web process in extended OWL-
  S,
- **Step 2**: we extend the specification with timing
  constraints,
- **Step 3**: we transform the system specification along
  with its imposed timing constraints into Timed Automata model,
- **Step 4**: we specify the formulae to verify and
  augment Timed Automata with a necessary
  variables as explained in section 5.2.
- **Step 5**: we automatically verify the formulae with
  Uppaal verifier,
- **Step 6**: If the property is not satisfied go to Step 1
to correct the OWL-S specification.

Figure 5. Model Checking Framework based on Timed OWL-S for
Web Service Time.

6.1. Transformation from Timed OWL-S to
Timed Automata

Timed OWL-S is taken to specify logical
structures, more important, timing constraints of
service processes with rich semantic information.
Transformed from the extended OWL-S process
model, Timed Automata model is constructed to
depict the structure and behaviour of the
(composite) service specified with service
operation and time semantic information. With
Uppaal tool, verification for temporal property
can be conducted. In the following, we describe
the transformation rules from Timed OWL-S to
Timed Automata

1. **Sequence**: The initial location is marked by
double circle.

   \[ \vdash \]

   Sequence(\( P_1, P_2 \))

2. **Choice**:

   \[ \vdash \]

   Choice(\( P_1, P_2 \))

3. **If-then-else**:
The International Arab Journal of Information Technology

If Cond then P₁ else P₂

4. **Split:** Network (set) of Timed Automata allows expressing the parallelism.

5. **Split-join:** Edges labeled with complementary actions over the common channel **Sync** synchronise.

6. **Repeat-While:**

7. **Repeat-Until:**

8. **Before:**

9. **Meets:**

10. **Overlaps:**

Since d₁ is not finished when d₂ starts, we use another clock y to count for d₂.

11. **Finishes:**

12. **Covers:**

13. **Starts:**

14. **Equals:**

6.2. **Discussion**

We use Uppaal simulator to prove the correctness of transformation rules. Let’s consider the overlaps case (Section 6.1 rule 10). We need computing the global time progress. For this purpose, we use an additional clock **Elapsed** reset when system starts i.e. \( x = 0, y = 0 \) and **Elapsed** = 0. For example, for overlaps(10,14), (12,20) see Figure 5, valuation of **Elapsed** clock at the final state should be 20. Effectively, Uppaal simulator indicates **Elapsed** ≥ 20.

\(^1\)Because of there is no invariant on final state, the system can reside infinitely.
Let's consider a wrong example overlaps ([10, 14], [16, 20]), the simulator indicates a deadlock. Thus, our approach allows us to avoid specification errors.

![Figure 5. Time Progress](image)

In the same manner, we have checked the correctness of all of the other transformation rules.

7. Case Study: Multimodal Transport

Multimodal transport is being used across a wide range of government, it is generally considered as the most efficient way of handling an international door to door transport operation. This is so because Multimodal Transport allows combining in one voyage the specific advantages of each mode, such as the flexibility of road haulage, the larger capacity of railways and the lower costs of water transport in the best possible fashion. Multimodal Transport also offers the shipper the possibility to rely on a single counterpart, the multimodal transport operator who is the architect of the entire journey and only responsible party from pickup to delivery, rather than having to deal with each and every modal specialist of the transport chain.

While Multimodal Transport seems to offer only benefits to all parties, shippers and service providers, it is also very difficult to achieve. Multimodal Transport requires a thorough control over all the steps involved in international transport; this means extensive use of information technologies that can provide freedom to plan and operate to carriers and reliable liability regimes to customers. On top of that Multimodal transport needs to be online tracking by the client; it is the fastest way to find out where the shipment is. It offers to the client real-time details of its shipment's progress via SMS or email.

In this case, we deal with the online shipment. TrackShip is a fictitious service that offers online tracking and helps customer to take an appropriate decision to change transport strategy when it is necessary. For example, the transport mode chosen may have to change over time when delays happen. So let's consider this hypothetical Scenario. The supposed itinerary combines sea and railway transport. The departure date is in 5 days from registration and the arrival is in 10 days. At arrival, if there is not an administrative problem, the railways transport is in 1 day and take 2 days to arrive at destination. The delay due to administrative problem may take 2 days. In this case, customer have the choice to change to air transport or keep the previous plan i.e. sea-railway combined transport. A confirmation must be done no later than 1 to 2 days after shipment arrives to port, confirmation interval must be finished at the same instant as the interval corresponding to the administrative problem processing is finished. The airport departure is in 2 hours from administrative problem solving and the air transit time is not more than 1 hour. Goods are in stock within 1 day from arrival.

In the following, we present a global description of Track-Ship service (Figure 6).

The Track-Ship service starts by login with AccName and ID, and after validate the NumberTracking, customer can track his own shipment by obtaining the following information: (1) ItineraryInfo: includes all necessary details about this itinerary i.e. departure, destination, transport mode, Duration, (2) Destination:The nearest call, (3) ElapsedTime, (4) EnabledOptions: depends on the delay happened due to administrative problem, new option is available to change the initial plan if required. The costumer can change the plan only and only if the precondition "AdministrativeDelay" holds, and which take 2 days.

![Figure 6. Track-Ship service diagram](image)

Now, for this case, we need to be sure that goods are in stock within fixed period of time. In the following, we try to verify the same property for different period of time.

7.1. Case A

Goods are in stock within 20 days and 3 hours (i.e. 483 hours) from registration.

7.1.1. Timed Automata Generation

By applying transformation rules presented in Section 6.1, we generate Timed Automata from the OWL-S specification. The generated Timed Automaton is

---

For reasons of simplicity, only a schematic diagram of Track-Ship service is presented.
presented in Figure 7. We can easily distinguish three main runs:

![Figure 7: Generated Timed Automaton](image)

- In orange, it corresponds to the situation where customer choose to change the transport strategy, to sea-air combined transport, in order to make up for lost time due to an administrative problem;
- In red, corresponds to the worst case where customer choose to keep the initial plan i.e. sea-railway combined transport in spite of delays.

7.1.2. Bounded Liveness Property Specification

![Figure 8: Augmented Timed Automaton](image)

In order to perform a model checking using Uppaal, we apply a set of transformations as explained in Section 5.2: the generated Timed Automaton is increased with the necessary variables i.e. a clock Elapsed and a boolean b. The formulae specification is \( \forall c(b \Rightarrow \text{Elapsed} \leq 483) \).

7.1.3. Uppaal Verifier Results

The verifier shows that the formula is satisfied (Figure 9. Hence, in all circumstances we respect predicted period.

7.2. Case B

Now, we take a shorter period. Goods are in stock within 19 days and 3 hours (i.e. 459 hours) from registration.

The generated Timed Automaton still the same as that shown in Figure 7. The formula specification is \( \forall c(b \Rightarrow \text{Elapsed} \leq 459) \).

7.2.1. Uppaal Verifier Results

The verifier shows that the formula is not satisfied as depicted in Figure 9.

![Figure 9: Bounded liveness properties verification](image)

7.2.2. Uppaal Simulator Results

In Figure 10, red edges are used to show run in question. It corresponds to the case when user chooses keeping the first plan in spite of delays. This run can be saved and traced by simulator.

![Figure 10: Run at the origin of the violation of the bounded liveness property](image)
condition \( \text{Elapsed} \leq 459 \). Notice that, the lower bound of the interval \([456, 480]\) corresponds to the arrival time at the airport and the upper bound correspond to the time in which goods is available in stock. For the other variables, the clock \( x \) is in \([0, 24]\), it counts the time in which goods should be in stock after arriving. The clock \( y \) is previously used corresponds to confirmation interval (i.e. \([24, 48]\)) which must be finished at the same instant as the interval corresponding to the administrative problem processing (i.e. \([0, 48]\)) is finished. Formally expressed as \( \text{Finishes} ([0, 48], [24, 48]) \).

![Variable valuations when goods arrive](image)

Figure11. Variable valuations when goods arrive

8. Conclusion and Future Work

This paper proposes a verification approach for bounded liveness property of Semantic Web service by Uppaal model checker. Taking account of time information, we use DAML-Time ontology in conjunction with OWL-S for describing the temporal content and the temporal properties of Web services then we transform the OWL-S specification to Timed Automata. So the verification process of service flow can be automated. The transformation rules proposed and a case study are presented to show applicability of our approach.

This research is still in the early stage, which only verification is automated, but the results are very encouraging. In future work, we aim to automate our proposed approach with a use of Model-Driven Engineering (MDE) tools for implementing transformation rules from Timed OWL-S ontology to Timed Automata model. In addition, we will extend the applicability of our approach to verify more properties. Finally, we also plan to verify more ambitious applications.

References


