Model Checking Web Services

Amit Shrigondekar, Lalindra De Silva, and Aravindan Thulasinathan

School of Computing,
University of Utah
{amitss, alnds, aravi}@cs.utah.edu
https://sites.google.com/site/modelcheckingwebservices/

Abstract. In today’s world where most of the business transactions are carried through web-based, online applications, it is essential to make sure the underlying models in these applications are correct and sound. Even more important is to verify if the interactions between these applications function according to preferred standards. With web services being the forefront of most of these web-based transactions, it is essential that we are able to validate a network of interacting web services and the individual services themselves to provide guarantees about their functions. Through this study, we make an effort to verify if a given set of web services abide by a set of desired properties and we validate if these properties would hold true. For this task, we make use of model checking which is a promising technique for verifying and validating software systems. We make use of a popular model checking tool used at present and show results on an airline reservation system which simulates the interaction of web services in the real world.

Keywords: Web Services, Uppaal, Model Checking, Airline Reservation Systems

1 Introduction

Web services are the leading communication mechanisms underlying a majority of the online business transactions that take place in the present world. A web service, defined to be a software system that supports machine-to-machine interaction over a network, has surpassed its original description and has facilitated a paradigm shift in the way people engage in commerce in the online world. Essentially, a web service is an autonomous, standards-based component and their public interfaces are defined using XML. A service provider enables the public to invoke this web service in a manner that is given by its definition (Fig. 1). Furthermore, most of the e-commerce processes, understandably, requires the interaction of many such web services. To provide much concrete examples, we can think of airline ticket reservation systems, online vendors such as amazon or ebay and even online banking that people take for granted without appreciating the role of web services that facilitate such tasks. In such examples, we can visualize how the aggregated web services arise as a requirement where several
web services, including travel agents, airlines, banks and credit card companies among many others, have to interact which each others’ web service.

However, it would be unwise to assume that every web service is guaranteed to behave the way they were modeled to be. This is more aggregated when multiple web services interact with each other and the chances of unexpected behavior is much higher than that of each individual web service. To take an example in the airline reservation systems, we can contemplate the possible scenarios where a failure of a single transaction in a particular web service - say the web service pertaining the credit card company involved - could have a detrimental effect on the rest of the web services and consequently the whole interaction. In addition to that, we can observe how constraints in each web service effect one another in this aggregated architecture. For example, there may be specific instances where time constraints are involved in a particular web service model and this in turn effects the rest of the services involved in the aggregated or composite web service architecture.

On a related topic, model checking has been an interesting advent in the field of formal verification and validation. Model checking had significant advantages over testing for the reason that the errors in models can be captured at the design stage itself and avoids the significant costs of testing once the systems are fully developed. In the setting of validating web services, the advantage of model checking in the early stages of the development can be highlighted. Also, model checking has been successfully applied to many case studies along similar lines and this motivates us to use model checking to validate the web services and the interactions within them.

Within this light, we make an effort to validate and verify an airline reservation model with additional services. For this task, we create our own models using the best approximation of real-world systems that do similar reservation tasks. It was noted, as we would discuss in a later section citing previous work, that most of these systems are based on web services and model checking them and their interactions has been of interest to some degree even before this work. We specifically introduce the notion of timed-automata in one of the sections in

**Fig. 1. Web Service Model**
these models as it was observed there are time constraints in some of the models involved in similar systems. Later, we evaluate our models using some of the LTL properties that we designed which we hoped would capture the correctness of the interactions involved.

In the remaining sections, we discuss our approach to model checking web services and we evaluate our study with the use of a concrete example of an airline reservation system. In section 2, we highlight a few previous research efforts in this area and study how previous researchers approached this problem. Section 3 describes Uppaal in detail. Uppaal [1] is the tool we utilized for this task and we emphasize on the applicability of Uppaal for similar tasks in this genre. We then discuss our models of the example that we are using for this study in section 4 and discuss the LTL properties we validated using our Uppaal on our models in section 5. We conclude the report with a discussion on some of the results and the lessons learned.

2 Related Work

Previous work along these lines have focused on three paradigms. Researchers have evaluated systems using 1) Pi-calcul model and verification, 2) Petri Nets model and verification and 3) Finite State Automata (FSA) methods and verification techniques. However, a few recent methods have focused more on using widely popular model checking tools for the same task. Wei et al. [2] uses SPIN, a widely known tool for model checking, to validate a composition of web services. They first translated the BPEL (a business process description language) definition of the web services into a FSA and used promela modeling language to model that FSA. They then used SPIN to validate and verify those processes written in BPEL.

Nakajima [3, 5] also carries out a similar study where he verifies an interaction of web services using SPIN. He designed several scenarios that could produce undesired behavior in such processes (including deadlocks) and used SPIN to verify those models. Another effort by Fu et al. [4] focused on a paradigm they define to be “conversation” which is the global sequence of messages exchanged by the peers in the web service composite. They show interesting results on what they define to be realizability properties that are conditioned on these conversations.

Diaz et al. [7] merged WS-CDL and BPEL and reported on a study in which they verified web services using Uppaal. They focused on timed-automata in setting which they may appear in such systems. On another study, Li et al. [6] produced what they called the IMWSC (Interaction Module for Web Service Composition) to verify composite web services and to validate their behavior.

3 Uppaal Model Checking Tool

Uppaal has become popular in the recent past as it provides a integrated tool environment for modeling, validating and verifying real-time systems. It’s a tool
jointly developed by the Department of Information Technology of Uppsala University, Sweden and the Department of Computer Science of Aalborg University, Denmark. Uppaal is most suited for modeling systems as a collection of non-deterministic processes with finite control structure and real-valued clocks. Being able to model timed-automata as such, is one of the main aspects of Uppaal. In systems where real-time controllers and communication protocols are critical, Uppaal tends to appear as an ideal tool for model checking (Fig. 2).

Furthermore, Uppaal has in built functionality that is common to most well-known model checking tools such as SPIN. Specifically, Uppaal has support for all the primitive data structures that are common to other prevalent tools as well as channel synchronization support and clock variables. Uppaal also has three main components built into it, namely, a description language, a simulator and a model checker. The description language is associated with an editor with which the modeling can be carried out in a visual form rather than coding it in. It enables the systems to be modeled as networks of automata augmenting them with the data types as necessary. The simulator provides a visual representation of the non-deterministic behavior of the models that are designed using the description language. It is augmented with a sequence chart as well as the ability to playback error traces as we encounter them.

The model checker enables us to check invariants and reachability properties by exploring the state-space of the system. Uppaal further provides different configuration options for exploring the state-space. It allows variations in search mechanisms (breadth-first, depth-first and random), conservative and aggressive
state-space reduction, different state-space exploration schemes as well as the ability to vary the hash-table sizes.

4 Model Description

For this study, we make use of an airline reservation system model and the complementary services that are usually involved in such a system. To provide a few examples of the web service interactions that we try to model, one can think of popular reservation websites such as Expedia or Kayak. Most of these websites interact with several parties, including the traveler, the agents for different services as well as the service providers themselves. Also, most of these applications enable the traveler (user) to reserve multiple things at any given point. For example, a traveler may reserve an airline ticket to a specific city as well as a room of a hotel in the same city.

Since most of the enterprise models for these web service based applications are proprietary, we made an effort to judge the underlying system to the best of our knowledge and to work backwards from the extracted information to model our own airline reservation system. We included car/hotel booking service models to the system to emulate the real-life scenario that goes behind most of these popular reservation websites. It is our contention that we put our best effort in modeling these systems as accurate, yet abstract, as possible so that they reflect the transaction models in an actual reservation system.

![Traveler interacting with airline, car and hotel booking servers](image)

The following are the models that we created using Uppaal for traveler, interacting with the airline server, car/hotel booking server through the agent
(Fig. 3). In the usual scenario, a traveler makes a request for an airline ticket through an agent and the agent communicates with the airline and other parties involved in financial aspects of this transaction, through web services. Especially, the information we gathered from previous work and from example models in the web revealed that, the agent communicates with its corresponding bank server as well as the airline server which is responsible for issuing tickets. The agent’s model is depicted in Fig. 4 whereas the airline model is in Fig. 5.
Also, we note that there is a timed-automata involved in reserving airline tickets. It was noticed that once the traveler makes a reservation, he/she has to book the tickets within 24 hours. This was modeled in Uppaal using a clock variable and this can be visually elaborated in the model as well. This is also explored using a sequence chart in Fig. 6.

![Sequence chart corresponding to the situation where a timer is enabled](image)

Fig. 6. Sequence chart corresponding to the situation where a timer is enabled

We added several other complementary services that most of these websites provide along with airline ticket reservation. Most of them are related to hotel rooms and cars. Thus, we modeled two separate sets of models for hotel room reservation and car reservation. However, these two sets of models are virtually the same, we discuss one set of models here (hotel room reservation) and leave the car reservation aside. In the hotel reservation model, the hotel agent commu-
Model Checking Web Services

communicates with the hotel server as well as the merchant bank. Here, the merchant bank refers to the bank corresponding to the agent. These models are described in Fig. 7 as well as in Fig. 8. The merchant model is given in Fig. 9.

![Fig. 7. Hotel server communicating with hotel agent](image1)

![Fig. 8. Hotel agent communicating with hotel and the merchant (hotel agent’s bank)](image2)

The next few models correspond to the payment scheme which involves credit card transactions. These models were generated by mining as much information as possible from the web as specific payment models are difficult to retrieve due to their proprietary nature. The payment process works as follows. When the merchant requests for information about the payment details, those details are transferred through the payment gateway to the payment processor. The
payment processor forwards that information to the card association (VISA, MASTERCARD) and the card association communicates with the user’s bank to complete the transaction. These details and the authorization messages are propagated back through the chain to the merchant who initiated the transaction and it is completed if everything succeeds. The payment gateway model is given in Fig. 10 and the payment processor is given in Fig. 11. The card association model and the issuing bank models are given in Fig. 12 and Fig. 13 respectively.

With the help of Uppaal, we are able to visualize a successful run of this entire payment scheme through the use of a sequence chart. This is displayed in Fig. 14.

5 LTL Property Verification

With these models in place, we made an effort to verify some of the properties that we assumed should be correct in the model we created. Uppaal has an in-built model checker that facilitates this and we provide a few examples of the
Fig. 11. Payment processor interacting with payment gateway and the card association

Fig. 12. The card association interacts with the payment processor and the issuing bank (user’s bank)

Fig. 13. Issuing bank interacts with the card association

properties we verified in the section below. The syntax for LTL properties in Uppaal correspond of the following.

- $E <> p$: There exists a path where $p$ eventually holds
- $A[ ] p$: For all paths $p$ always holds
- $E[ ] p$: There exists a path where $p$ always holds
Fig. 14. A successful run of the payment process involved

- \( A \leftrightarrow p \): For all paths \( p \) will eventually hold
- \( p \rightarrow q \): Whenever \( p \) holds \( q \) will eventually hold

Following are a few of the interesting properties we verified using these models. A complete list of LTL properties that we tried out are listed in the project website.

- There exists a path where airline booking is available and the time is \(<24\) hours.
  
  \[ E[\ ][\ ]^{\downarrow} \text{airline.booking\_available} \text{ and } x < 24 - \text{[Satisfied]} \]

- The traveler should be able to cancel the booking after reservation
  
  \[ E[\ ][\ ]^{\downarrow} \text{airline.reservation\_timer\_start} \rightarrow \text{traveler.cancel\_booking} - \text{[Satisfied]} \]

- There should be a path for the traveler to change his/her itinerary once it’s available
• $E[\square] \text{traveler.trip\_available} \rightarrow \text{traveler.check\_trip}$ - [Satisfied]

– If the traveler cancels a booking, it should always return to the starting position

• $A \leftrightarrow \text{traveler.cancel\_booking} \rightarrow \text{traveler.start}$ - [Satisfied]

– If money was withdrawn from traveler’s bank, it should imply that the traveler has received the ticket

• $E[\square] \text{issuing\_bank\_money\_withdrawn\_from\_account} \rightarrow \text{traveler.tkt\_received}$ - [Satisfied]

6 Contributions of Team Members

The following are the contributions to this project by each of the team members.

Amit Shrigondekar:
– Designing the models for airline, travel agent and traveler
– Maintaining website, updating and adding information regularly
– Initial study of previous research work on this area
– Contributed in designing LTL properties

Lalindra De Silva:
– Initial surveying and integration of Uppaal tool
– Designed models for hotel/hotel agent and contributed to LTL properties
– Assisted in final report writing

Aravindan Thulasinathan
– Designing models for car/car agent and designing LTL properties for them
– Assisted in final report generation
– Contributed in the initial survey of research study

7 Discussion

We have shown in this work how we can model check web services to assure the correct behavior when they are interacting with each other. The models that we generated are valid and this is evidenced by the properties that we validated using Uppaal. We also emphasize that the models reflect real-life scenarios where we have exhaustively tried our best to model them according to the limited knowledge we could gather from the web and from previous work.
We also highlight the efficiency of Uppaal as a tool in similar projects. Uppaal proved to be a very handy tool that could ideally fit in such scenarios where the systems have to be modeled as networks of (timed)automata. The versatility of the tool was obvious throughout this project and it proved much easier to use than contemporary tools, such as SPIN, that we would have used otherwise.

References

5. Nakajima, Shin *On verifying Web service flows* Applications and the Internet (SAINT) Workshops, 2002
7. Gregorio Daz and M. Emilia Cambronero and Juan J. Pardo and Valentín Valero and O Cuartero *Model Checking Techniques applied to the design of Web Services* 2007