Analyzing Web Services Interaction

Using Open ECAT Nets

Latreche Fateh
Sebih Hacene
Belala Faiza
ANALYZING WEB SERVICE INTERACTION USING OPEN ECATNETS

Latreche Fateh, Sebih Hacene, and Belala Faiza

LIRE Laboratory, University of Constantine, Route de Ain El Bey, Constantine, Algeria

Abstract:

Current technology and description languages related to the SOA (Service Oriented Architecture) paradigm give limited support to formally analyze web service compositions. Consequently, several works have been done to tackle and investigate the composition process. In this work, we propose using Open ECATNets, a sound combination of Open nets and ECATNets (Open Extended Concurrent Algebraic Term Nets) to model compactly and soundly the internal logic and message exchange behavior among peer services. Thanks to this model we do not only obtain a high level specification of service choreographies, but we are also able to formally reason on it.

Keywords: Interaction Analysis, Web Services, Open ECATNets.

1. INTRODUCTION

Nowadays, web services and other web-based applications are the fitted technical solution to implement Service Oriented Architecture (SOA). The advent of this new paradigm has brought to the fore new dimensions, such as quality of service and robustness to withstand inevitable faults that require revisiting established component-based concepts in order to meet the new requirements of the service-oriented paradigm. Current technologies based on Universal Description, Discovery, and Integration (UDDI), Web Service Description Language (WSDL), and Simple Object Access Protocol (SOAP) do not offer rigorous solutions to these requirements, in particular they provide a limited support in service composition and its analysis.

Some efforts of both researchers and industrial practitioners have given birth to several new programming and description languages customized to the specification of web service composition such as: WSCL (Web Service Conversation Language), BPML (Business Process Modeling Language), XLANG (XML business process language), WSFL (Web Services Flow Language), BPEL (Business Process Execution Language for Web Services), WS-CDL (Web services choreography description language), and WSCI (Web Service Choreography Interface).

However, all aforementioned technologies and languages have been initially proposed to provide a syntactical solution that still requires a semantic part; they described web services by using XML syntax, which lacks both a well defined semantic and sufficient expressive power to release the vision of diverse web services having wide-scale interoperability [13]. Consequently, several groups have proposed formal semantics for some existing web services languages.

Among these attempts, there are some based on finite state machines, process algebras and Petri Nets [6], [8]. Thus, formal methods have provided a foundation for web based software by successfully addressing challenging issues such as mathematical models for service specification, composition and adaptation, or rigorous approaches for verification, deployment, testing, and certification.

Petri nets models through their extensions have been largely used for this purpose. They give a graphical web services models suitable for analysis and simulation using several existing tools. In particular, Open Petri nets [12], a refinement of Petri nets with interface places for communication and initial and final designated markings, have been also proposed to tackle special issues in web service choreography composition.

Our contribution is closely related to these approaches, it provides a richer and more adapted semantic framework for the services interaction specification allowing diverse analysis methods of computer-aided verification. In this paper, we propose Open ECATNets (Open Extended Concurrent Algebraic Term Nets) as a special case of ECATNet formalism proposed initially by Bettag and Maouche [2]. They allow modeling web service composition involving complex data structures in a compact way, they benefit also from their formal definition in term of rewriting logic.

This work objective is to develop a new web services specification approach allowing integrating in a unique framework, the formal specification, analysis and the flexibility at the execution time (run-time). We propose using Open ECATNets to analyze interaction of service choreographies; more precisely we exploit
the reflective capability of rewriting logic to reason correctly about concurrent conversation among peers’ services. Besides, as far as dataflow aspect is considered, semantic of our model is not restricted to a particular composition language.

The rewriting logic has been introduced by José Meseguer [9], as a consequence of his work on general logics to describe concurrent systems. It allows correct reasoning on concurrent system behaviors having states and evolving in terms of transitions. This logic benefits also from the presence of numerous tools and operational environments such as the Maude system [3] which is at the same time an expressive language.

The remainder of this paper is organized as follows. Section 2 will give an overview of related works. Section 3 introduces rewriting logic via its practical language Maude and ECATNets model. Section 4 proposes the rewriting logic based semantic model for Open ECATNets. Section 5 describes our main contribution showing how to give web services interaction model a formal semantic using Maude functional and system modules. The derived semantics model is then used by the Maude LTL model checker to prove some functional properties that are inherent to dynamic composition of web services. Syntactic and semantic concepts introduced in this work will be illustrated via a simple case study. The paper is outlined by some concluding remarks and perspectives.

2. RELATED WORK

Several attempts to formalize web service behavior have been proposed. The most emerging approach is the Petri nets based one since it preserves the behavioral of individual web service semantics and offers panoply of analysis tools. Petri nets variants are also used to take care of more various aspects of coupling services such as: semantic compatibility of exchanged messages, non-functional compatibility and behavioral compatibility.

However, all emerging Petri nets based approaches do not tackle all the above aspects of services composition at once. For instance, in [15] an executable architectural description language called WS-NET was proposed to facilitate the verification and monitoring of web services. This language integrates Colored Petri Net semantics with the object-oriented concepts to verify and stimulate service composition. Also, Hamadi et al propose in [5] a Petri net based algebra, to model only control flows that are considered as fundamental concept in reliable web service composition process.

An hierarchical CPNet (Colored Petri Net) model has been used in analysis and verification of web services composition [14]. Authors of this work propose a set of translation rules between composition language and hierarchical CPNet, and then CPNet analysis tools are used. This work is very promising but it requires a formal analysis to guarantee the correct transformation between different models.

In [8] and [12] authors have respectively proposed Open Workflow Net (oWFN) and Open Nets as a formal foundation to specify services and their interaction. Their work, related to some novel activities and constructs of WS-BPEL 2.0, was focused on the explicit modeling of the interface. This allows analyzing only the communicational behavior of a service without considering an appropriate semantics of the dataflow aspect. Moreover, unlike our approach, all Petri nets based works are founded on hybrid models and tackle formalization of only some concepts which are closed to the control flow of web services interaction.

Another category of works exists abstracting away the notion of semantics and exhibit different analysis executions via special kinds of tools, ranging from simulators to model checker. Thus, they tempt to formalize web service composition while transforming it to other models that come with some convenient tools. For example in [10], WSFL descriptions are translated into Promela (the input language of the SPIN model checker) in order to analyze composite web service.

Besides, the authors of [11] proposed an analysis tool called WSAT (based on the SPIN model checker) that accepts as input a BPEL4WS description and some LTL properties. In this kind of works user requires to specify both system and formal properties that the system should satisfy.

Our proposal is quite different since our objective is raising reliability of services composition by analyzing interaction among peer’s services. Only one formalism is used, rewriting logic (RL), to specify syntactic aspects of web services and dynamic ones [7]. The presented RL-based semantics is feature-complete; that is, it models all data and control flow aspects of web services composition in a compact way. Besides, the true concurrency semantics which is the main feature of this logic may be exploited to model service instances, and message correlation, not yet covered by the operational semantics of Petri nets model. Thus, links of service partners evolving dynamically can be naturally specified and analyzed.

3. BASIC CONCEPTS

In addition to the use of the rewriting logic (RL) in specification of concurrent and distributed system semantics, it is also a promising logical framework in which many logics and concurrency models (such as ECATNets) can be naturally represented and interrelated [3]. The objective of this section is to present elementary concepts of
rewriting logic and ECATNets model allowing a good comprehension of this work.

### 3.1 Rewriting Logic and Maude

Rewriting logic is logic of change that allows expression of concurrent and nondeterministic computation in a very suitable manner. In this logic, static aspect of systems is represented by membership equational theories and dynamic aspect is represented by rewriting theories describing the possible transitions between states of concurrent systems.

Equational theories allow modular specifications of systems; they are multi sorted theories in which basic statements are either equations of algebraic terms or memberships.

Rewriting theories extend equational ones by adding a set of rewriting rules. Each rewrite theory \( T \) is a 4-tuple \( (\Sigma, \text{EUA}, \varphi, R) \), where \( (\Sigma, \text{EUA}) \) is the equational theory describing states of the system, \( \varphi \) is the function specifying the frozen arguments of each operator in \( \Sigma \) and \( R \) is a set of rewriting rules (noted \( [t] \rightarrow [t'] \)) modeling the possible transitions between states of the concurrent system. \([t] \) and \([t']\) are equivalence classes of algebraic terms belonging to the set \( \Sigma/\text{EUA} \).

Reasoning in rewriting logic is accomplished by finite application of the following rules:

1. **Reflexivity.** For each term \([t] \) in \( \Sigma/\text{EUA} \):

   \[ [t] \rightarrow [t] \]

2. **Congruence.** For each \( t \in \Sigma_n, n \in \mathbb{N} \):

   \[ [t_1] \rightarrow [t'_1], ..., [t_n] \rightarrow [t'_n] \Rightarrow [f(t_1, ..., t_n)] \rightarrow [f(t'_1, ..., t'_n)] \]

3. **Replacement.** For each rewriting rule:

   \[ r : [t(x_1, ..., x_n)] \rightarrow [t'(x_1, ..., x_n)] \in R \]

   \[ [w_1] \rightarrow [w'_1], ..., [w_n] \rightarrow [w'_n] \]

   \[ [t'(w'/x)] \rightarrow [t'(w'/x)] \]

4. **Transitivity.**

   \[ [t_1] \rightarrow [t_2], [t_2] \rightarrow [t_3] \Rightarrow [t_1] \rightarrow [t_3] \]

Rewriting logic is also a reflexive logic, i.e. aspects of its meta-theory can be represented in a consistent way, namely there is a universal theory \( U \) in which any finitely presented rewrite theory \( R \) (including \( U \) itself) can be presented as a term \( \overline{R} \), any terms \( t, t' \) in \( R \) as terms \( \overline{t}, \overline{t'} \), and any pair \( \overline{R}, \overline{t} \) as a term \( \overline{R}, \overline{t} \), so that the following equivalence is established:

\[ R \models t \rightarrow t' \Leftrightarrow U \models \overline{R}, \overline{t} \rightarrow \overline{R}, \overline{t'} \]

Rewriting logic has also many operational environments, the most known is the Maude language. Maude is a declarative language based on rewriting logic, used as a meta-language to create different environments. It regroups three types of modules mainly: functional modules to define the static aspects of a system, they form a Maude sub-language (extension of OBJ3) based on the equational logic; system modules specify the dynamic aspect of the system using rewriting rules; while object oriented modules specify the objects oriented systems.

The fact that specifications in rewriting logic are executable makes possible to have a flexible formal model of system which can constitute a prototype for the analysis and validation phase.

In particular, the Maude system [3] offers a powerful model checker (LTL) for checking systems properties. It acts as follows: it takes as input a system model (the module "M") expressed in rewriting logic formalism, and a specification (the module "M-Preds") which represents a system specification property expressed in linear temporal logic. For a given initial state of the system (expressed in the module "M-Check"), it performs a calculus using the "on the fly" local methods principle to produce two possible results. Positive result if all the model executions satisfy the specification and negative result if at least one execution of the model does not satisfy the specification. In this case, the Model-Checker gives this execution or it’s a simplification as a counter example which serves for the user to correct the source of the problem and then re-execute a new checking of the model.

### 3.2 ECATNets

ECATNets are high level algebraic Petri nets initially proposed by Bettaz and Maouche [2]. They combine the expression power of Petri nets and abstract data types. By using ECATNets we do not only gain highly condensed system model, but we reach also an attractive theoretical model according to their simplicity and intrinsic concurrent nature. In an ECATNet, transitions, places and arcs are labeled by elements of the multiset (noted as: \( \Sigma/\text{EUA}(X) \)) of algebraic terms belonging to an intended algebra \( \Sigma/\text{EUA}(X) \).

**Definition 1:** An ECATNet (see Figure 1) is a high level Petri net having the structure \( E = (P, T, \text{sort}, \text{IC}, \text{DT}, \text{CT}, \text{TC}, M_0) \), where:

- \( P \) and \( T \) are respectively finite set of places and transitions (with \( P \cap T = \emptyset \)),
- \( \text{sort} : P \rightarrow S \) is a function that associates to each place an algebraic sort \( s \) of \( \Sigma \),
- \( \text{IC} : P \times T \rightarrow \Sigma/\text{EUA}(X) \) is a function that specifies partial conditions on input place markings, \( IC(p,t) \) may have the following forms: \( IC(p,t)=a^0 \) (marking of \( p \) must be equal to \( a \), an algebraic multiset terms), \( IC(p,t)=a' \) (the algebraic multiset terms \( a \) must be included in marking of \( p \)) and finally \( IC(p,t)=\bot \) (the algebraic multiset \( a \) must not be included in marking of \( p \)),
- \( \text{DT} : P \times T \rightarrow \Sigma/\text{EUA}(X) \) is another function that associates to each input arc
(p×t) of transition t, a multiset of algebraic tokens to be consumed from input place (Destroyed Tokens),

- \( CT : P \times T \rightarrow MT_{ECAT}(X) \) associates to each output place of P, a multiset of algebraic terms (Created Tokens) which may be added to when a transition is fired and
- \( TC \) is an additional condition, when it is omitted, the default value is the term true.
- \( M_0 : P \rightarrow MT_{ECAT}(0) \) constitutes the initial marking of ECATNet

\[ p : (s) \]
\[ IC \ (p,t) \]
\[ DT \ (p,t) \]
\[ t \]
\[ TC(t) \]
\[ CT(p',t) \]
\[ p' : (s') \]

Figure 1: Graphical Representation of ECATNet

The originality of ECATNets resides in their behavior semantics given in terms of rewriting logic [2], [1].

### 3.3 OPEN ECATNets

Our main objective in this work is to propose an ECATNet based model to verify correctness properties of web services composition. We define Open ECATNets to model compactly and soundly the internal logic and message exchange behavior of service choreographies.

An Open ECATNet is a particular case of ECATNet with input places, for receiving information, and output places for emitting information. Thus, mapping Open ECATNets concepts to web service ones is straightforward. First, a web service behavior which is basically a partial ordered set of operations may be naturally represented by an Open ECATNet. Web service operations and states are respectively modeled by ECATNet transitions and places. Similarly, arrows between places and transitions in ECATNet are used to specify the causal relation between Web services operations. Moreover, the use of algebraic terms tokens allows modeling data context of web services taking into account their semantic compatibility.

**Definition 2:** An Open ECATNet \( (E, IO, ST, M_0, M_f) \) is a particular ECATNet \( E \) having the following additional structure:

- \( IO \) is a set of (input/output) places having a well defined sort,
- \( ST \) is a set of state places having the sort Bool (for Boolean values), \( IO \cup ST = P \)
- \( M_0 \) and \( M_f \) are respectively the initial and final marking.

Thus an Open ECATNet is a particular case of an ECATNet with: a set of interface places (IO) designed to external interaction, a set of state places (ST) to model state of service partners and finally an initial and a final marking (M0 and Mf).

The formal semantic of Open ECATNet is expressed in terms of rewriting logic. Thus, any service, represented by an Open ECATNet is defined as a rewriting theory (see figure 5). This will facilitate its extension to model new web service composition concepts, and will make easy its interconnection with services of other applications in the context of heterogeneous systems. Also, unlike Open nets and Open Workflow Net, time can be straightforwardly modeled thanks to Real Time extension of Maude.

Furthermore, checking Open ECATNets specifications will also be possible thanks to Maude analysis tool. In particular, finite state service choreographies can be analyzed by using the LTL Maude model checker. Regarding services composition with infinite states which still not analyzable by Open Nets implementation [12], using rewriting logic via its language Maude will allow their analysis by using infinite-state equational abstraction technique.

In this section we show through the following Maude modules how static and dynamic aspects of Open ECATNets can be defined in rewriting logic in order to reason correctly on them.

In fact, we have adopted a modular approach in which a set of Maude modules (rewriting theories implementation) are used (see Figures 2 to 5). The proposed model consists in: i) presenting tokens abstract data types by the functional module “TOKENS”, ii) presenting marking structure by the functional module “PTNSPEC”, iii) presenting marking of interface places by the module “INTERFACE” and finally, iv) each service partner will be presented by a system module extending the interface module, in such module marking and transitions are formalized respectively by means of one equation and a set of rewriting rules modeling send and receive operations.

```maude
fmod TOKENS is
  //tokens sorts and subsort relations
  sorts Srt1 ... Srtn Tokens .
  subsorts Srt1 ... Srtn Bool < Tokens .
  //tokens values
  ops t1 ... tn : -> Srt1 [ctor] .
  ...
  ops tn1 ... tnn : -> Srtn [ctor] .
  //empty token value
  op none : -> Tokens [ctor] .
endfm
```

Figure 2: Open ECATNet Tokens

```maude
fmod PTNSPEC is
  //importing the TOKENS module
  protecting TOKENS .
  sorts Place Marking Markings .
  subsorts Marking < Markings .
  //defining marking of a place
  op <_,_> : Place Tokens -> Marking [ctor prec 21] .
  //joining places marking
  // empty marking value
  op none : -> Marking [ctor] .
endfm
```

Figure 3: Open ECATNet Marking Structure
fmod INTERFACE is
  //importing PTNSPEC module
  including PTNSPEC.
  sort Interface .
  subsort Interface < Markings .
  op Interface : -> Place [ctor] .
  //fixing the interface marking
  eq interface = <Pi1,none> ... <Pin,none>.
endfm

Figure 4: The Generic Module INTERFACE

The proposed approach is general enough since all aforementioned modules represent a generic semantic model for an Open ECATNet based web service. Thus, they remain valid for any web service example. Each service type is generated through the operation of a peer name declaration (Figure 5). Thus, service instances are naturally designed according to this modeling approach.

mod SERVICE is
  including Interface .
  //peer name declaration
  op ServName : -> Markings [ctor] .
  //state places
  op ps1 ... psn : -> Place [ctor] .
  //variable declarations ...
  //fixing the initial marking of a service
  eq ServName = <ps1,true> ... <psn,false> .
  //send and receive operations
  rl [!send]:<ps1,true> <ps2,false> <Pi1,none> M => <ps1,false> <ps2,true> <Pi1,message> M.
  rl [?receive]:<ps1,true> <ps2,false> <Pi1,message> M => <ps1,false> <ps2,true> <Pi1,none> M .
endm

Figure 5: Open ECATNet as a Maude System Module

Open ECATNet model seems to be very suitable to specify complex web service behavior that may be defined by the direct pairwise composition of the involved Open ECATNets specifying individual web services. This formal composition results in a new Open ECATNet. Thus, recurrent composition allows composing more than two Open ECATNets. Also, both dataflow and control flow of web service interaction are well defined on the characterization of such concepts in terms of syntactic and dynamic constructs of rewrite theories. Besides, thanks to subsort relation interface places with messages having common features are now merged in a single place. This provides a more compact representation.

Moreover, this formalization can be used with slight modifications to release analysis with the Maude analysis toolkit. In subsequent section we show how using model-checker tool to analyze some correctness properties like disabled activity, dead activity and deadlock state in the context of web services choreographies.

4. CASE STUDY

In order to illustrate our proposal concerning the specification of both dataflow and control flow of web service interaction, we consider the Virtual Travel Agency choreography (Figure 6). The objective of this composite web service is to provide a flight booking service to his travelers.

In this composition, the “User” service launches the process by sending a reservation request (“! flight request”) stating his destination and the period of time to the “Agent” service. Then, this last decides whether or not this request can be fulfilled. So, the “User” service will then receive either an offer message (“? Offer”) or a not available notification (“? na”). If reservation offer is received, the User can accept (“! ack”) or reject it (“! nack”) by sending the corresponding message to the Agent service. In case of positive answer the flight is successfully booked and the reservation ticket is sent, otherwise the “User” service can formulate a new request.

Although this composition is a simple one, complex interactions amongst participating services are produced. Using an Open ECATNets for their analysis will facilitate errors fixing.

4.1 SPECIFICATION STEP

In this section we show how Open ECATNet can be used to formally describe and execute the Virtual Travel Agency choreography.

So, in order to generate the Open ECATNet specification for the Virtual Travel Agency composition, the first thing we need to do is to declare two new system Maude modules “USER” and “AGENT” (Figures 7 and 8) extending the module “INTERFACE”, and they will contain web services names declarations (User, Agent) and web services state places (p1, ..., p6, p1',..., p6'). Then, we add to these two modules a set of rewriting rules to mechanize
transitions. Indeed, only one equation is included in each module to specify clearly and in a global manner each web service marking; this represents a typical instance of an Open ECATNets marking.

As a final point, to make previously declared peers able to communicate through interface places, we define the system module “CompWs” (Figure 9), in which we include “USER” and “AGENT” modules, and then we combine marking of peers’ definitions and interface marking by means of one equation.

Through the presented modules of this section, we achieve a modular and legible specification of composite Web service. In the same way this specification can be easily enriched, particularly, we can add other elements to specify new concepts. In particular, to manage instantiation and dynamic replacement of services designer can use reflection mechanism and internal strategies. Also, instances of the same interface structure (“Interface” marking) may be used by several interactions of web services. Indeed shared places are attached with various arcs, labeled differently. Semantic compatibility of exchanged data is guaranteed thanks to sort and subsort concepts. Indeed marking by means of one equation.

In this section we show how this tool can be used in order to analyze interaction between peers services modeled as Open ECATNet. In fact, we have combined in this work meta-level capability of Maude with the model-checker tool; this allows to reason correctly about peers interactions modeled as rewriting rules. More precisely, we have used the descent function “metaApply” to check whether or not a transition is fireable at a given state. So, in order to check our Virtual Travel Agency choreography, we have defined the module “WS-PREDS” (Figure 10). In this module, we import the module “CompWs” to be analyzed as well as the three predefined module: “META-LEVEL”, “SATISFACTION” and “MODEL-CHECKER”, then we define a set of useful atomic propositions.

In this paper we are interested by model checking behavioral problems of service compositions. For the first property (Figure 11), we have checked whether or not the sending flight request transition (“flight request!”) is a dead activity. Using the proposition “Is Dead” for a given applied rewriting rule captured in the Meta level context, model execution shows that the property is not preserved; we get then a useful counter example showing why this property fails.

In the second correctness property example (Figure 11), we check whenever a flight request occurs; we eventually receive an offer or not available notification. This complex property expression will be possible thanks to “Enabled” proposition declaration and some LTL operators (<> and V).

Similarly, the third command in Figure 11 tests that every ticket sent by the Agent service will be subsequently received by the user peer.
Finally, we have checked the existence of deadlock states which are not a final state. This may be achieved thanks to another type of proposition: “Deadlock” and “Final Marking”.

mod WS-PREDS is
  protecting CompWs.
  protecting META-LEVEL.
  including SATISFACTION.
  including MODEL-CHECKER.
  subsort Markings < State.
  op Markingof_in : Place Markings -> Tokens.
  ops Enabled disabled IsDead IsLive : Qid -> Prop.
  ops FinalMarking Deadlock : -> Prop.
  var C : Markings.
  var Q : Qid.
  var P : Prop.
  var p : Place.
  var tk : Tokens.
  eq Markingof p in <p,tk> C= tk.
//test whether or not the rule labeled Q is enabled at a given state or no
  eq C|=Enabled(Q)=
    (metaApply(upModule('CompWs,false), upTerm(C), Q, none, 0) /= failure).
//test whether the rule labeled Q is disabled at a given state or no
  eq C|=disabled(Q)=
    (metaApply(upModule('CompWs,false), upTerm(C),Q,none,0)==failure).
//test if the actual marking is a final one
  eq C|=FinalMarking=((Markingof p5 in C==true )
    and (Markingof p5’in C==true)).
//test if the rule with label Q is dead, i.e the rule Q is disabled in all states
  eq C|=IsDead(Q)=(modelCheck(C,[]
    disabled(Q))==true).
//check deadlock state; i.e a state with all rules disabled
  eq C|=Deadlock=(modelCheck(C, disabled('! flight request)) and modelCheck(C,
    disabled('!na)) and ..........
and modelCheck(C, disabled('!flight ticket)).
  eq C |= P = false [owise].
endm

Figure 10: The Module WS-PREDS

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>reduce in WS-CHECK :</td>
<td>modelCheck(cmpws, IsDead('!flight request)) .</td>
<td></td>
</tr>
<tr>
<td>rewrites: 27 in 878'711'567'9ms cpu (5ms real) (0 rewrites/second)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result Bool: true</td>
<td>-----------------------------</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>reduce in WS-CHECK :</td>
<td>modelCheck(cmpws, [](Enabled('!flight request) -&gt; &lt;&gt;</td>
<td></td>
</tr>
<tr>
<td>rewrites: 57507 in 118'378'796'50ms cpu (7'86ms real) (0 rewrites/second)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result Bool: true</td>
<td>-----------------------------</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>reduce in WS-CHECK :</td>
<td>modelCheck(cmpws, [](Enabled('!flight ticket) -&gt; O Enabled('? flight ticket)) .</td>
<td></td>
</tr>
</tbody>
</table>

Figure 11: Web Service Choreography Properties Analysis

5. CONCLUSION

The scope for using formal methods in the area of web services is wide, and the challenges raised by this new area can offer opportunities for applying these formal techniques to model more structured and complex services.

Our investigation has shown the advantages of using Open ECATNet model, for web services specification and composition. The proposed Open ECATNets model allows capturing relevant and significant information of such systems in a compact way. Besides, the use of a formal based rewrite theories model allows natural specification of individual services concurrent behaviour and message exchanged. The verification of properties and the detection of inconsistencies both within and between services are also possible. Hence, we have taken advantages of practical tools developed around rewriting logic framework. For instance, composing web services that satisfy requirements such as security through interaction and which is a challenging issue, will be naturally tackled in this approach.

In our future works we plan to focus on extending Open ECATNet to model other dynamic web services concepts like runtime events and quality of service. Such extension will allow us to consider behavioural adaptation.

On the analysis aspect, we intend to verify infinite-State service choreographies by using abstraction mechanisms offered by Maude toolkit.

REFERENCES:


