Mop-ECATNets for Dynamic Web Services Formal Modeling

FATEH LATRECHE
LIRE laboratory, Constantine 2 University
Constantine, Algeria
fateh_lat@yahoo.fr

FAIZA BELALA
LIRE laboratory, Constantine 2 University
Constantine, Algeria
faiza.belala@univ-constantine2.dz

Abstract – Mop-ECATNets are a sound combination of Meta Petri Nets and OpenECATNets (Open Extended Concurrent Algebraic Term Nets). They inherit fl exibility of control from Meta nets and data structure, concurrency and composability from Open-ECATNets. Our aim in this work is to show how Mop-ECATNet, a layered Petri nets based model, makes it possible the formal execution and analysis of dynamic web services at design time. We argue that it is possible and valuable to provide a real-time rewriting logic based model that accounts for failure events and recovery actions. In order to illustrate its potential, we apply our approach through a detailed case study of a traveler map dynamic service.

Keywords – Dynamic Web services, Open-ECATNets, Meta Petri nets, real-time rewriting logic

1. INTRODUCTION

Service Oriented Architecture (SOA) is a development model aiming at raising integration and interoperability of software components. In SOA services are units of work, they are self-describing modular applications that can be published, localized and invoked across the web thanks to a set of XML standards. In most cases elementary services cannot carry out required functionalities. Thus, we need invoking several Web services and then composing them in order to reach new personalized, rich and more interesting functionalities.

Several efforts on web service modeling and analysis have already been done by researchers and industrial practitioners. But, mostly these works pay less attention to dynamic features. Also, current Web service standards and languages provide limited constructs for specifying exceptional behavior [1] [2], and they are semi-formal requiring translation to generate formal models that admit checking.

In this work, we propose using Mop-ECATNets (Meta Open Extended Concurrent Algebraic Term Nets) to enhance design and analysis of dynamic Web services. Mop-ECATNets are an extended version of Open-ECATNets [3], allowing the formal specification of failure events and recovery actions.

Since their introduction, by Carl Adam Petri [4] in the beginning of sixties, Petri nets have been successfully applied through their extensions as a foundation for software systems addressing many challenging issues. In particular, Meta Petri nets [5] as multi-level Petri net model have been proposed for modeling dynamic processes and their control. Also, Open-ECATNets have been proposed to tackle various aspects of Web service compositions, such as semantic compatibility of exchanged messages, functional compatibility and service refinement. Open-ECATNets benefit from their definition in term of rewriting logic.

Rewriting logic is a general framework, in which not just applications, but entire formalisms and computation models can be naturally expressed. This logic allows correct reasoning on concurrent systems behavior. Several practical languages were invented on the basis of rewriting logic, the most known is Maude (SRI laboratory, United States)[6], a declarative language where several concurrent applications have been considered. Furthermore, various extensions of rewriting logic and Maude were made in order to provide additional representation and reasoning abilities. Real-Time Maude is one of these extensions [7], used for specifying and analyzing real-time and dynamic systems.

The remainder of this paper is organized as follows. In section 2, we give an overview of related works. Sections 3 and 4 introduce the real-time rewriting logic and Open-ECATNets model. In Section 5, we deal with the Mop-ECATNets model for dynamic Web services. We identify its integration in real-time rewriting logic semantic framework, so both structural and behavior aspects are considered. In section 6, we illustrate our formalization approach by means of a realistic case study, the traveller map dynamic service. We also show how some relevant properties are formally checked. The paper is outlined by some concluding remarks and perspectives.

2. RELATED WORK

Several attempts [08] [09] [10] have already been conducted to formalize web services but, without taking into account their dynamic features such as, changes of user profile, invocation context and quality of services. In this section, we situate our work regarding some theoretical and architectural models dealing with dynamic issue of web services.

Authors of [11] have given a Petri net based methodology for context aware services design. Thus, context aware Web service is defined by a Petri net, where places represent sensed/inferred context concepts and transitions represent the actions. The advantage of this work is the well description of context aware applications and its possible integration within dynamic environment thanks to contextual information. But, this approach makes only use of first-order logic predicate formalism to specify context concepts. This makes dynamic web services models quite restrictive.

In a similar work [12], Dmytro Zhovtobryukh defines also a context aware Web service framework composed of two stages: planning and composition reconfiguration. According to this work, a context aware Web service is defined as a complex functional unit having an explicit inputs and outputs; it is also able to perform two or more mutually exclusive service functions selected by a context function. In this work a Meta transitional Petri net having two layers is used. The lower layer contains the service net and the control layer contains one or more Meta control nets. This proposal is very promising, but it suggests using ordinary Petri nets in both basic and control layers. This remains inappropriate for modeling real-time and dynamic constraints. Also, the proposed composition approach is lacking implementation details.

Our approach for specifying and analyzing dynamic Web services is quite different, since it uses useful features of many Petri nets models and it's based on a unique semantic framework: real-time rewriting logic. Mop-ECATNets are also executable; they are able to reconfigure dynamically their structure. Thus, services failures are conditions that trigger adaptation actions. Besides, our developed model allows formal checking of some inherent properties related to dynamic Web services thanks to the Real Time Maude model checker tool.

3. REAL-TIME REWRITING LOGIC

The objective of this section is to present elementary concepts of real-time rewriting logic for more details, reader can refer to [6] [7] [13].

Since its introduction, rewriting logic has attracted the interest of both theorists and practitioners. Its experimentation through many applications has conducted to propose some extensions of this logic. The most significant one is the real-time rewriting logic allowing specification of real-time systems. Basic specification elements of real-time rewriting logic are real-time rewrite theories [14]. Rewriting rules of a real-time rewrite theory are of two kinds: I_R and T_R, I_R is a set of instantaneous rewrite rules, and T_R is a set of tick rewrite rules.

Real-Time Maude is an extension of Maude. It emphasizes ease and generality of specification, including support for distributed real-time object based systems. Also, it provides a large range of simulation and analysis techniques. This tool complements other real-time analysis tools not only by the full generality of the specification language and the range of analysis techniques, but also by its simplicity and clarity. Real-time and hybrid systems are represented as real-time rewrite theories, where equations are used to specify data types and rewrite rules to mechanize dynamic and real-time behavior.

The main form of formal analysis consists in simulating system behaviors by using the timed rewriting commands. Besides, to gain more assurance about a system, designer can use timed search and timed linear temporal logic model checking to explore many concurrent behaviors of a system [14].

4. OPEN-ECATNets PRESENTATION

In addition to the use of rewriting logic in specification of concurrent and distributed system semantics, it is also a promising logical framework in which many logics and concurrency models as ECATNets [15] may be naturally represented and interrelated. ECATNets are high level algebraic Petri nets initially proposed by Bettaz and Maouche [15]. They combine 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 multi-set (noted as: $MT_{\Sigma/EUA}(X)$) of algebraic terms belonging to a given algebra $T_{\Sigma/EUA}(X)$.

Definition 1. An ECATNet (see Fig. 1) is a high level Petri net having the structure *(P, T, sort, IC, DT, CT, TC)*, where:

- *P* and *T* are respectively finite sets of places and transitions (with $P \cap T = \emptyset$);
- *sort:* $P \rightarrow S$ is a function that associates to each place an algebraic sort *s* of *Σ*;
- *IC* (Input Condition):$P\times T \rightarrow MT_{\Sigma/EUA}(X)$, is a function that specifies partial conditions on input place markings;
- *DT* (Destroyed Tokens): $P\times T \rightarrow MT_{\Sigma/EUA}(X)$, is another function that associates to each input arc *(p×t)* of transition t, a multi-set of algebraic terms to be consumed from input place;
- *CT* (Created Tokens): $P\times T \rightarrow MT_{\Sigma/EUA}(X)$ associates to each output place of *P*, a multiset of algebraic terms which may be added when a transition is fired;
- *TC* (Transition Condition) is an additional condition, when it is omitted, the default value is the term true.

A distinctive characteristic of ECATNets is their strength in modeling synchronization constraints and complex system data structures. They have been largely applied in modeling and checking various kinds of systems [16] [17] [18].

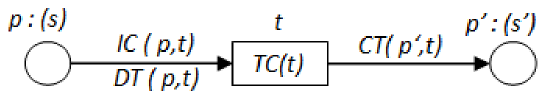

Figure 1 Graphical representation of ECATNet

To make ECATNets able to model open systems capable to interact with their environment, we have, in a previous work [3], enriched their model by additional structures, inspired from the open net model [10]. In particular, Open-ECATNets are endowed with some open places representing the service interaction points with environment. Formally their definition is as follows:

Definition 2. An Open-ECATNet : {E, Mi, Mf} is a particular marked ECATNet E : (P, T, sort, IC, DT, CT, TC), such that:

- P = IO U ST, where IO is a set of (Input/Output) places having a well defined sort and ST is a set of state places,
- Mi and Mf are respectively the set of initial and the final markings such that: $\forall$ p $\in$ IO, Mi(p)= Mf(p)=0.

In an Open-ECATNet, the set of interface places (IO) are designed to define external interaction and the set of state places (ST) to model services states.

Open-ECATNet (or ECATNet) behavior is defined by way of concurrent computations in rewriting logic. Each transition is materialized by a local rewriting rule of a given rewrite theory defining the distributed structure of ECATNet states.

5. OPEN-ECATNets EXTENSION

Because of their size and complexity, distributed and dynamic systems, as Web services, may not be easily modeled. They require using more tailored formalisms. Dynamic web services are highly affected by time. A rigorous approach to tackle failures is to fix a maximal waiting time before launching adaptation process. The objective of this section is to present the Mop-ECATNets model, designated to better specify and analyze dynamic Web services.

To allow a flexible modeling of complex dynamic Web service process, we enhance Open-ECATNets with the Meta Petri nets features [5]. In Meta Petri nets places, transitions, links and tokens of a lower level are active (available), if and only if their meta place contains at least one token.

The Mop-ECATNets model is based on an extension of two level Meta transitional nets where Meta places control lower level transitions. More precisely, each transition (send or receive action) of the lower level is active regarding tokens of Mop-ECATNets lower level (exchanged messages), if and only if the transition meta place contains the same token. Mop-ECATNets are also equipped with timing constraints: On one hand, all lower level transitions are forced to fire when they are enabled; firing in this case is an atomic action. On the other hand, some state places are constrained by timestamp. This allows specifying the requests adaptation timers (i.e. modeling the fact that requested functionalities may be failed if elapsed time after the sending request action is equal to a fixed maximum delay). Formally, the Mop-ECATNet definition is given by:

Definition 3. A Mop-ECATNet ME =< E, O, Q, λ > is a two levels Meta transitional net having as higher level net the marked ECATNet E and as lower level the Open-ECATNet O. Q is the incidence function mapping meta places of E to transitions of O and λ : ST → N is a function that maps each state place (belonging to ST) to its reconfiguration timer.

Let's recall that: E = (P', T', sort', IC', DT', CT', TC') and O = {(IO∪ST, T, sort, IC, DT, CT, TC), Mi, Mf}. In Fig. 2, we give a Mop-ECATNet, where places mpt and mpt' are the meta places relative to transitions t and t' of the lower Open-ECATNet. The Mop-ECATNet of this figure can also be defined by the following sets: P'= {mpt, mpt' }, T'= {mt}, Q = {(mpt, t), (mpt', t')}, IO = {pi, pi' }, ST = {p1, p2, p3},Mi = {(1, 0, 0)} , Mf = {(0, 1, 0), (0, 0, 1)} and λ = {(p3, 3)}.

5.1. Mop-ECATNets Semantics

Our formalization of a Mop-ECATNet is based on the following idea: a net marking is considered as a multi-set of algebraic terms modeling places and its transitions as a multi-set of rewriting rules. In order to mechanize instantaneous firing of transitions, we add a timer for each transition t belonging to T U T'. This timer has either the value 0 to denote forced to fire transitions, or the value INF (infinity) for non firable transitions.

Thus, Mop-ECATNets are defined in terms of timed rewrite theories. Three timed Maude modules TOKENS, MARKING-SPEC and

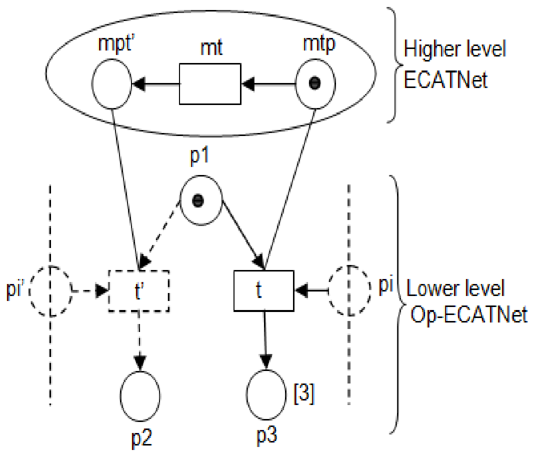

Figure 2 A Mop-ECATNet example

ACTIV-TRANS (Fig. 3 to Fig. 5) are designed to implement our Mop-ECATNets model. The Figure 3 shows a slightly modified version of the timed Maude module TOKENS that represent tokens abstract data types. In this module, we firstly import the predefined module NAT-TIME-DOMAIN-WITH-INF defining the time domain to be the natural numbers with the infinity value. Then, we declare a set of sort and subsort relations useful to describe statically Mop-ECATNet tokens (a multi-set of algebraic terms having the form tkid, tkid[t], where tkid is the token identifier and t represent the container place reconfiguration timer).

Mop-ECATNets Marking is then formalized by the timed module MARKING-SPEC (Fig. 4). According to our formalization, markings are viewed as multi-sets of ordinary places, Meta places, ordinary transitions and Meta transitions (operations declared in lines 6, 7, 8 and 9 of Fig. 4). Let's note that in the case of transition, its algebraic term must have the form < t, r >, where t is the name of transition and r is its timer value.

The third timed module ACTIV-TRANS (Fig. 5) defines, thanks to a set of equations, in which case a given transition is active regarding a particular token (the send or receive action is enabled for a particular exchanged message).

For Mop-ECATNets model, it is worth noting that formalized services may be extended to model new Web service composition concepts, and will make easy their interconnection with services of other applications in the context of heterogeneous systems. Also, time which is currently an important aspect of Web services, is straightforwardly modeled thanks to real-time extension of Maude. Furthermore, checking

Mop-ECATNets specifications is now possible thanks to the Real Time Maude analysis tools.

```
tmod TOKENS is
including NAT-TIME-DOMAIN-WITH-INF.
sorts Token Token.
subsort Token Nat < Tokens.
subsort TkId < Token.
 op _`[_`] : TkId Time -> Token [ctor prec 21].
op __ : Tokens Tokens -> Tokens [ctor assoc comm prec 22
id: none ].
op none : -> Tokens [ctor].
endtm
```

Figure 3 Tokens of a Mop-ECATNet

```
tmod MARKING-SPEC is
including TOKENS.
sorts OrdPlaceNm OrdTransitionNm MetPlaceNm
PlaceName TransitionName  MetTransitionNm Marking .
subsorts OrdPlaceNm MetPlaceNm < PlaceName.
subsorts OrdTransitionNm  MetTransitionNm <
TransitionName.
op <_,_>:OrdPlaceNm Tokens->Marking[ctor prec23].
op <_,_>:MetPlaceNm Tokens->Marking[ctor prec23].
op <_,_>:OrdTransitionNmTimeInf->Marking[ctor prec 23].
op <_,_>:MetTransitionNm TimeInf->Marking[ctor prec23].
op __:Marking Marking->Marking[ctor assoc comm
prec24 id:none].
op none:-> Marking [ctor].
 op [_]:Marking-> System [ctor].
endtm
```

Figure 4 Mop-ECATNets Marking as a timed module

```
tmod ACTIV-TRANS is
including MARKING-SPEC .
op Active : OrdTransitionNm TkId Marking -> Bool .
var tk : TkId .                    var tks : Tokens .              var
ti : TimeInf .
ops t1 ... tn  : -> OrdTransitionNm [ctor] .
 ops mpt1 .... Mptn : -> MetPlaceNm [ctor] .
eq Active( t1 , tk , < mpt1 , tk tks > M:Marking )  = true .
...              ...             ...
eq Active( tn , tk  , < mptn , tk tks >  M:Marking ) = true.
eq Active( t:OrdTransitionNm , tk  ,  M:Marking ) = false
[owise]  .
endtm
```

Figure 5 Active transitions in a Mop-ECATNet

5.2. Mop-ECATNets and Dynamic Web Service

Dynamic behavior of Mop-ECATNets may be defined independently of the specified system, but in this work we explain its motivation through Web services components (see Fig. 6). In this case, five generic rewriting rules define instantaneous behavior of Mop-ECATNets. These rules model firing of Mop-ECATNets transitions when their timers have the value 0. As results of firing, some tokens are removed from definite places and added to other places, transition timers are turned off (fixed to the infinity value), and the function *recomputeTimers* is applied to the entire resulting state to recompute all transition timer values. Particularly, this function tests if a transition is enabled, then its timer is reinitialized to the value 0, otherwise the timer is left unchanged. More precisely, for each transition of a generic Mop-ECATNet (see Fig. 6) we associate a rewriting rule having the following interpretation:

- The rewriting rule associated to send-req transition, models how to send requests. By firing this rule a request token is added to the interface place pi1 and the same token, but constrained by a reconfiguration timer τ , is added to the subsequent state place p12 of the requester (The timer τ represent the maximal waiting duration for a response),
- Rewriting rule that fires the *receiv-req* transition models how to receive requests. When applying this rule, a request token is consumed from the interface place *pi1*, the initial marking of the place *p21* (denoting the maximum number of possible requests) is decreased by one, and a token is added to the subsequent state place *p22* of the requested service,
- The sending reply rewriting rule associated to transition *send-reply* is executed when treatment duration of a *p22* tokens decreases to the value 0, i.e. the request has been treated, by executing this rule a reply token is sent to the interface place *pi2*,
- A conditional rewriting rule is also associated to the transition *receiv-reply*, it models how to receive a response from a requested Web service. This rule is applied when token treatment duration is lower than reconfiguration Timer value of the same token,
- Adaptation process is handled by the Meta transition *reconf* mechanized by a conditional rewriting rule; this conditional rule is triggered when token reconfiguration Timer expires and there is no response token in the interface place *pi2*. By triggering this rule, we get a new marking of the higher net component.

Through the proposed definitions of this section, we have achieved a modular and legible specification of dynamic Web service compositions. The expressiveness and generality of real-time rewrite theories allow us the declaration of both user defined operators and real-time dynamic behavior. Another important fact of the rewriting Mop-ECATNet implementation is that each deduced Maude module specifies not just a theory, but a precise high-level mathematical model. Hence, this model will serve to the formal checking of any dynamic web services based system. It is also worth to mention that the MopECATNet definition presented in this section can serve also in modeling other systems with temporal constraints and having dynamic and adaptive behaviour.

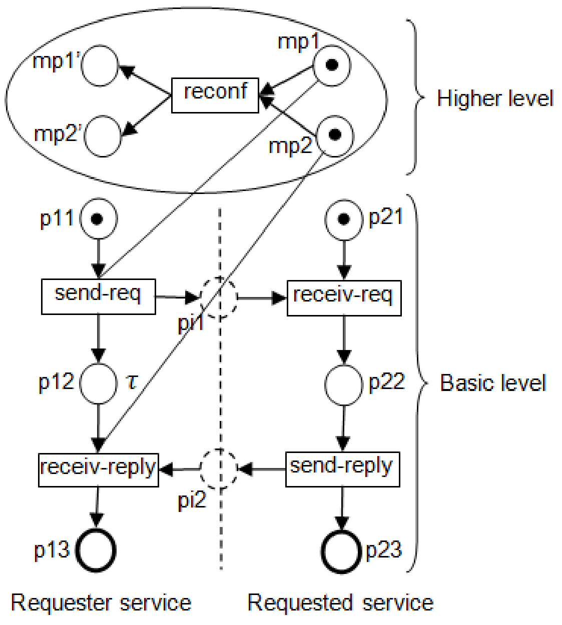

Figure 6 Mop-ECATNets dynamic behavior

6. CASE STUDY

In order to illustrate our proposed model, we consider in this section the traveller map web service as case study (Fig. 7). This composite service is initially composed of two services: a mobile map requester and a primary map provider service, and can be adapted at reconfiguration time by integrating a secondary map service provider in case of failure.

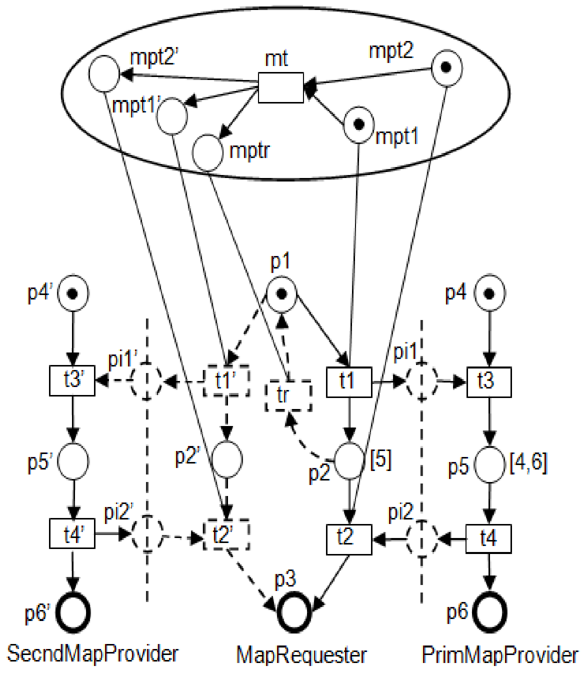

Figure 7 The traveller map web service

6.1. The specification step

We note that our proposed model is generic enough, it remains valid for any Web service example. So, in order to formalize the traveller map composite web service on the basis of Mop-ECATNets model, we need to declare a new real-time Maude module extending MARKING-SPEC and ACTIV-TRANS modules (presented in section 3) by additional constant operators to specify: the Traveller map dynamic service components identifiers, ordinary places names, ordinary transitions name, meta-places name, meta-transitions name and finally tokens identifier. Dynamic behavior of our case study is automated by a set of rewriting rules and their effects on the corresponding Mop-ECATNet elements.

6.2. Analysis of the traveller map service

In order to check our traveller map dynamic service we define the module MODEL-CHECK-TRAV-MAP-BEHAV (Fig. 8). In this module, we import the traveler map dynamic behavior module TRAV-MAP-BEHAV (not presented here due to lack of space), as well as the predefined module TIMED-MODEL-CHECKER. Then, we define some useful atomic parameterized propositions (equations of the module in Fig. 8):

1. The atomic proposition delay-elapsed is used to define tokens reconfiguration timer expiration;
2. *reconf-start* which checks triggering of adaptation process for a request;
3. *success* that confirms accomplishment of a request.

For this paper, we restrict our self to verify three compound properties (Fig. 9). The first timed command checks whether the delay expiration in time less or equal to four time units is possible. The second timed command checks if delay expiration for the request appears, it will be always followed by the triggering of adaptation process. Finally, we check absence of successful treatment in time less or equal to four time units. The result of these commands is showed in fig. 9.

```
(tmod MODEL-CHECK-TRAV-MAP-BEHAV is
including TIMED-MODEL-CHECKER .
 protecting TRAV-MAP-BEHAV.
vars tks tks' tks'' : Tokens .   var tk : TkId .
var SYSTEM : System .  var REST : Marking .
op delay-elapsed  : TkId -> Prop [ctor] .
op reconf-start : TkId -> Prop [ctor] .
op success : TkId -> Prop [ctor] .
eq {[ < p2, tk [ 0 ] tks > REST ]} |= delay-elapsed(tk) = true .
eq {[ < mpt1', tk tks > < mpt2',tk tks' > < mtr, tk tks'' > REST ]} |=
reconf-start(tk) = true .
eq {[ < p3 , tk tks' > REST ]} |= success(tk) = true . endtm)
```

Figure 8 The module MODEL-CHECK-TRAV-MAP-BEHAV

```
Model check{TravMS}  |=t <> delay-elapsed(tk:TkId) in
MODEL-CHECK-DYN-TRAVLER-SERV
 in time <= 4 with mode default time increase 1
Result[ModelCheckResult]:
counterexample({{[< mpt1,tk1 tk2 tk3 > ..in time 0,'t1}{{[<...in
time 0,'t3}{{[ in time 0,
unlabeled}{{[... in time 0,'t1... in time 0,'t3}{{[... in time
0,unlabeled}{{[< ....]} in time 4,'tick})
===============
Model check{TravMS}  |=t[](delay-elapsed(tk:TkId)=> <>
reconf-start(tk)) in
   MODEL-CHECK-DYN-TRAVLER-SERV in time <= 6 with mode
default time increase 1
Result Bool :   true
=================
Model check{TravMS}  |=t[]~ success(tk:TkId)in MODEL-
CHECK-DYN-TRAVLER-SERV
in time   <= 4 with mode default time increase 1
Result[ModelCheckResult]:
counterexample({{[< mpt1,tk1 > < mpt1',(none).Tokens > <
mpt2,tk1 > < mpt2',(
 none).Tokens > ... < tr,INF >]} in time 4,   'tick})
```

Figure 9 Examples of the traveller map dynamic service properties analysis

In this section, we have illustrated by using the traveller map dynamic service how to use our model. More precisely, we have showed how a Mop-ECATNet based specification may be formally analyzed against dynamic system requirements. Obtained results confirm usefulness of this model, it takes into account many aspects of dynamic systems. In fact, Mop-ECATNets have the ability to model systems having concurrent, timed, dynamic and reconfigurable behavior also dataflow aspect is considered. Rewriting logic is unifying logical and semantic framework; this implies that Mop-ECATNets can be linked easily with other formalisms.

7. CONCLUSION

In this work, we have introduced Mop-ECATNets as a new Petri net based model, devoted to raise reliability of dynamic Web services design and re alization. The proposed model inherits a set of useful constructs from other Petri net models and allows developer to reason correctly about dynamic features at design time thanks to its integration in timed rewriting logic. We have shown how this logic provides a flexible conceptual framework, where Mop-ECATNets can be naturally seen as a timed rewrite theory. A nice consequence of this axiomatization is that relationship between the two defined Petri net levels, Metalevel for reconfiguration process and basic one for service model, was formally established. In addition dynamic properties of a Mop-ECATNet were checked and deduced, in particular traveller map dynamic service case study has been used to illustrate our model and check some useful properties.

In the proposed model, services are adapted by substitution caused by the time expiration. As future works, we will focus on modeling other adaption triggering events like users' profile changes and appearance of more adjusted services.

8. References

[1] O. Ezenwoye, S. Busi, and S. M. Sadjadi, "Dynamically reconfigurable data-intensive service composition," in WEBIST (1), pp. 125–130, 2010.

[2] D.H. Xu, Y. Qi, D. Hou, Y. Chen, and L. Liu, "A formal model for dynamic web services composition mas-based and simple security analysis using spi calculus," in Proceedings of the Third International Conference on Next Generation Web Services Practices,

NWESP ’07, (Washington, DC, USA), pp. 69–72, IEEE Computer Society, 2007

[3] F. Latreche, H. Sebih, and F. Belala, “Analyzing web service interaction using open ecatnets,” in ACIT2011 International Conference on Information and Technology, 2011.

[4] C. A. Petri, Kommunikation mit Automaten. PhD thesis, Darmstadt University of Technology,Germany, 1962.

[5] V. Savolainen and V. Terziyan, “Metapetrinets for controlling complex and dynamic processes,” 1999.

[6] J. Meseguer, “Conditional rewriting logic as a unified model of concurrency,” Theor. Comput.Sci., vol. 96, pp. 73–155, Apr. 1992.

[7] P. C. Olveczky, Real-Time Maude 2.3 Manual, 2007. http://www.ifi.uio.no//RealTimeMaude/.

[8] R.Hamadi, and B. Benatallah, “A Petri Net-based Model for Web Service Composition”. In: ADC, 17, pp. 191-200, 2003.

[9] N. Lohmann, P. Massuthe, C. Stahl and D. Weinberg, “Analyzing Interacting WS-BPEL Processes”, in Proc. Int. Conf. Business Process Management, pp. 17–32, 2006.

[10] W. M. van der Aalst, A. J. Mooil, C. Stahl and K. Wolf, “Service Interaction: Patterns, Formalization, and Analysis,” In: Formal Methods for Web Services - 9th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Web Services, LNCS, vol. 5569, 2009.

[11] T. Lu and J. Bao, “A systematic approach to context aware service design,” JCP, vol. 7, no. 1,pp. 207–217, 2012.

[12] D. Zhovtobryukh, Context-aware Web Service Composition. Jyvaskyla studies in computing,University of Jyvaskyla, 2006.

[13] . C. Olveczky and J. Meseguer, “Semantics and pragmatics of real-time maude,” Higher Order Symbol. Comput., vol. 20, pp. 161–196, June 2007.

[14] K. Bae and P. C. Olveczky, “Extending the real-time maude semantics of ptolemy to hierarchical de models,” in RTRTS (P. C. Ölveczky, ed.), vol. 36 of EPTCS, pp. 46–66, 2010.

[15] M. Bettaz and M. Maouche, “How to specify non-determinism and true concurrency with algebraic term nets,” in Selected papers from the 8th Workshop on Specification of Abstract Data Types Joint with the 3rd COMPASS Workshop on Recent Trends in Data Type Specification, (London, UK, UK), pp. 164–180, Springer-Verlag, 1993.

[16] M. Bettaz, M. Maouche, and K. Barkaoui, “Formal specification of communication protocols with object-based ecatnets.,” in EUROMICRO, pp. 492–, IEEE Computer Society, 1996.

[17] A. Hicheur, K. Barkaoui, and N. Boudiaf, “Modeling workflows with recursive ecatnets,” in Proceedings of the Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC ’06, (Washington, DC, USA), pp. 389–398, IEEE Computer Society, 2006.

[18] N. Boudiaf and A. Chaoui, “Double reduction of ada-ecatnet representation using rewriting logic,” Enformatika Journal (Transactions on Engineering, Computing and Technology), vol. 15, pp. 278–284, October 2006.