

Temporal Exception in Web Service Composition

Bey Fella, Bouyakoub Samia, Belkhir Abdelkader
 Institute of Electronics & Computer Sciences
 USTHB University
 Algiers, Algeria
 Bey-Fella@hotmail.com
 belkhir@lsi-usthb.dz
 bouyakoub.s@gmail.com

Abstract— Web service composition is studied by many works and constitutes the heart of a great research activity. However, the majority of this work does not take into account temporal exception handling. Consequently, the results do not answer the needs and the temporal preferences of customers and suppliers. Incorporating temporal constraints in Web service composition results in a more complex model and addresses the crucial need for exception handling. In this paper, we present H-Service-Net model for Web service composition and policies of handling exceptions. We validated our proposed approach in an implementation called H-Service-Editor tool.

Keywords— *Composition of Web services; Petri network; temporal constraints; handling of exception.*

I. INTRODUCTION

Due to its capability for dynamic composition and easy reuse, Service Oriented Architecture (SOA) has become a popular framework for software development in many application domains. An important process in SOA is service composition [9].

A major part of the interest created by so-called Web services is their possibility to distribute processing capabilities across a number of loosely coupled functional entities communicating over standardized messages [6]. Companies such as Google, Amazon or PayPal have been making an increasing part of their revenue by exposing and selling their functionalities as instances of services, reusable as background components by third-party application developers [6].

A composite Web service invokes one or more other Web services and combines their functionality. In contrast, a Web service that does not invoke other Web services is called a basic Web service. The process of developing a composite Web service is referred to as a Web service composition [10].

The objective of Web services composition is to determine a combination of services according to the customer's request. This composition will seem to the customer as a single service because it is transparent to him. In composition, Web services collaborate by exchanging messages. In addition to this exchange, other factors affect this composition. We are interested in the time factor which is crucial and at the same time very complex.

Due to failures that can occur using Web services and their composition, several solutions of exception handling

have been proposed in order to recover from these exceptions.

Exceptions are critical in Web services. Therefore, it is essential to take into account the handling of exceptions, especially if their execution relates to the continuation of the composed service. The rest of this paper is organized as follows. Section II defines some related work. Section III presents our model called H-Service-Net. Section IV addresses the policies of handling exception with a demonstrative example. Section V presents the H-service-editor tool, its architecture and usage. Conclusions close the article.

II. RELATED WORK

In the literature, several theories have been proposed to explain exception handling, which is a critical case in Web services, therefore, to have a consistent execution, it is essential to take into account exceptions. In service composition it is important to define the mechanism of handling exceptions in order to have a coherent and consistent composition even in the presence of exception.

A number of approaches have been proposed to deal with exception handling in Web service composition. First, Caoqing et al. [1] describe the integration model which contains the normal process and exception handling logic based on Petri net integration technologies. The result shows that this model can realize formal modeling of exception handling in Business Process Execution Language (BPEL), and further provides support for analysis and verification of properties relating to exception handling. A novel architecture for exception handling has been proposed by Thirumaran et al. [2] for focusing upon and verifying the manageability of a web service. But this solution does not provide methods of handling exceptions, it only detects if an exception is manageable or not. The approach proposed by Wang et al. [3] describes the policy-based exception handling approach for BPEL processes, which is a new framework designed for exception handling in BPEL processes to provide a flexible language mechanism. The approach of Erradi et al. [5] proposes a set of extensible recovery policies to declaratively specify how to handle and recover from exception in Web services composition. The identified constructs were incorporated into a lightweight service management middleware named Manageable and Adaptive Service Composition (MASC) to transparently enact the fault management policies and facilitate the

monitoring. Hamadi et al. [4] propose Self-Adapting Recovery Net (SARN), an extended Petri net model, for specifying exceptional behavior in business processes. SARN adapts the structure of the underlying Petri net at run time to handle exceptions. The approach of Christos et al. [7] introduces the Service Relevance and Replacement Framework (SRRF) whose main concept is resolving exceptions by finding relevant Web tasks, exploiting qualitative and functional metadata semantics. Finally, Lau et al. [8] propose an approach to server-side exception handling by composite Web services that capture Unavailability and Time-out exceptions and provide Retry as recovery action, or Throw to propagate exceptions.

Exception handling is generally tedious and error prone. The issue of exception handling has not been carefully considered in existing service composition works [9].

Although several studies have indicated that exception handling is critical, little attention has been given to time out exception.








III. H-SERVICE-NET: A TEMPORAL MODEL FOR WEB SERVICE COMPOSITION

The H-Service-Net model (an acronym for Hierarchical Service Net) is a time Petri net-based model. It allows the modeling of time-critical aspects in the field of Web Services. It allows incremental composition of services, as well as consistency checking after each modification. It introduces a new type of places named composite places. A composite place is an abstract place represented by a sub-network, allowing a degree of independence between the parts of the H-Service-Net. Indeed, a composite or single place in H-Service-Net depends only on the subnet to which it belongs. In other words, the modification of a component can affect its subnet or the subnets of the same hierarchy. This representation allows for incremental modeling of the H-Service-Net. This will allow for easy correction of errors, an exact location of conflicts between the subnet elements and support rapid changes. Thus, the H-Service-Net model is well suited for modeling the synchronization constraints in a temporal scenario. As a result, it was chosen to model the composition of Web services. We present in what follows the different elements of the H-Service-Net model:

A. Places in H-Service-Net

There are two main types of places of H-Service-Net: simple and composite." Then, Table I shows places in H-SERVICE-NET, simple place are modeled by circle shape and composite place are modeled by triangle shape.

TABLE I. PLACES IN H-SERVICE-NET




Place	Modeling	Description
Ordinary Place		It models a basic element (Web service) and its execution time.
Virtual Place		It models a temporal constraint.
Silent Place		It models a place without any specific task, which is used to handle exceptions.
Parallel Place		It models a set of elements of the same group that operate simultaneously, which is used to model concurrent Web service.
Sequential Place		It models a sequence of elements of the same group that run in sequence. It is used to model sequential Web services.
Root Place		It represents the root of the global Petri net and behaves like a sequential composite place.
Loop Place		It is an element that runs in a loop and is used to model a recursive Web service.

An H-Service-Net can be seen as a tree where the parallel and sequential composite places represent intermediate nodes, and atomic places associated with Web services are the leaves.

B. Transitions in H-Service-Net

An operation performed by a Web service is modeled by a transition. Table II shows transition in H-SERVICE-NET that define the different termination semantics :

TABLE II. TRANSITIONS IN H-SERVICE-NET

Transition	Modeling	Description
Simple transition		It is fired when all its input places are active and have available tokens.
Master transition		It is fired as soon as the place associated with the Master arc is active and has an available token.
First transition		It is fired when one of its input places is active and has a free token.

C. Tokens and arcs in H-Service-Net

H-Service-Net defines state tokens and exception tokens and there are two types of arcs in the model, then Table III shows tokens and arcs in H-Service-Net:

TABLE III. TOKENS AND ARCS IN H-SERVICE-NET

Arc \ token	Modeling	Description
State Token	●	It defines the state of the Web service associated with the place
Exception Token	▲	It is used to handle exceptions
Simple arc	→	Control the firing of a simple transition
Master arc	→	Control the firing of a Master transition.

The modeling of Web services in H-Service-Net is given as follows:

- A simple or atomic Web service is represented by an ordinary simple place.
- A composite Web service is represented by a sequential, parallel or loop composite place.

D. Example of the TimeOut policy:

Let us consider the following scenario:

A person wants to go on holiday, and wants to make a campsite for two days in a forest at the end of the holidays. Inclusive vacations are offered by a travel agency that can offer a composition of the following Web services for this request:

- Ws1, Ws6: two Web services of payment by credit card.
- Ws2: Visa Service.
- Ws3: Sales service ticket travel agency.
- Ws4, Ws5: two Services of booking rooms in two different hotels.

Figure 1 shows the time constraints (date) in the network H-Service-Net of the example.

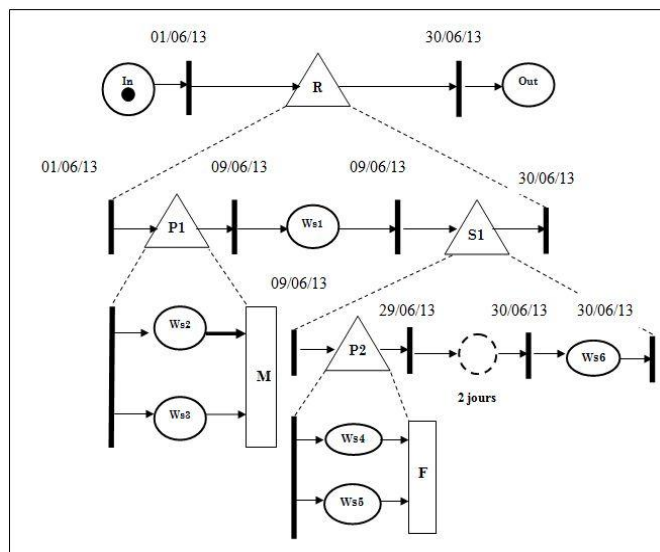


Figure 1. Temporal Constraints in H-Service-Net for the example.

For the modeling of these Web Services in a single H-Service-Net, we add the following composite places:

- The visa services ws2 and the sales service ticket travel agency ws4 can run in parallel, so we represent them in H-Service-Net by a parallel composite place P1.
- The Web Services of booking rooms ws4 and ws5 can run in parallel, so they are modeled in H-Service-Net by a parallel composite place P2.
- The composite services P2 and Web Payment service banking ws6 are modeled with the sequential composite place S1.
- The Composite services P1 and S1, and the Web Service Payment by credit card ws1 run in sequence, so we have modeled this set by the root place R.

IV. SOLUTION TO TEMPORAL VIOLATIONS BY A HANDLING OF EXCEPTION

Exception handling is critical in Web services composition. It is essential to consider exception handling for the robustness of the composition, especially if the exception relates to the continuation of the composition.

In this section, we present the policies of exception handling in H-Service-Net. For each exception, a set of methods is executed to handle the exception. We were inspired by Rachid et al. [4] to define the methods of exception handling. Table IV shows all used methods:

TABLE IV. BASIC OPERATIONS FOR EXCEPTION HANDLING

Method	Description
CreatePlace(nameWS, type-place, type-transitionIn, type-transitionOut)	Allows creating a simple or composed place with an input transition and output transition.
CreateSilentPlace(nameSilentPlace, type-transitionIn, type-transitionOut)	Allows creating a silent place without any task between two types of transition.
AddExceptionToken(place)	Allows adding an exception token to a silent place.
DisableWS(ws)	Allows canceling the running Web service.
AddTo(P, ws)	Allows to add the place ws to the composed place P
AddBefore(ws, ws')	Allows adding the place ws' before the place ws.
AddAfter(ws, ws')	Allows adding the place ws' after the place ws.
ReplacePlace(ws, ws')	An existing place ws is replaced by another place ws'.
RemoveToken(ws)	A token of state is removed from the Web service ws.

A. The TimeOut policy:

In a Web service composition, both the client and the service provider may have temporal constraints in their interaction. For example, the service provider may cancel the service if it does not receive a response after a time T_{max} and similarly for the service requester.

To handle the time out exception, each Web service is associated with a time limit T_{max} . If the execution of a Web service exceeds this time limit, a time-out exception is triggered.

A policy of handling TimeOut (WebServiceWsTO, time T) exception is defined in Figure 2:

-
- PSEUDO CODE 1.** The TimeOut policy
1. DisableWs(WsTO). //Cancels the Web service which reached the timeout.
 2. Compensate (WsTO) or RollBack (WsTO). //Optional
 3. RepeatAfter(WsTO). //Repeat the Web service which has reached the timeout (defined in the next section).
 4. Compensate (WsTO) or Rollback (WsTO). // Optional.
 5. RepeatAfter(WsTO, 2*T)// If another exception of time out is raised, Repeat the Web service till the doubling timeout expired.
 6. Compensate(WsTO) or Rollback (WsTO). // Optional
 7. OtherWS(TimeOut, WsTO, OtherWsTO). //If another exception of time out is raised, search another alternative service (defined in the next section).
-

Figure 2. Pseudocode of the TimeOut policy.

In order to handle the time out exception, the Web service which has reached a timeout is cancelled. Then, the Web service is repeated using the RepeatAfter policy and if another timeout exception is captured Repeat the Web service until the doubling timeout expired and expect that the service will not miss this deadline with $T = 2 * T$ (by doubling the time out). If this strategy does not work, the exception can be handled by calling another equivalent and compatible Web service.

Compensate(WsTO) : undoes the effect of task execution result of the Web service (WsTO).

In compensation, information must be added in the SOAP header in order to define that the Web service can be compensated or not, `<nsto :compensate enable= "True" />` or `<nsto :compensate enable= "False" />`.

Rollback functionality RollBack (WsTO) is executed in the reverse order of their forward execution and rolled back to (WsTO) original state.

The difference between compensate and rollback is that compensate is a Forward Recovery and rollback is a Backward Recovery, compensation or rollback are not require in read-only Web service.

Compensation is run on the service provider's side by a call to service. In our solution, compensation is considered as a service call cancellation

If the temporal constraint assigned to the Web service is higher than the execution time of the timeout policy (RepeatAfter (Ws, T) + RepeatAfter (Ws, 2*T)) then all the timeout policy is executed. Otherwise if the temporal constraint is higher than the execution time of the RepeatAfter (Ws, T) policy then only the stage one, two, three, four and seven are executed in order to satisfy the

temporal constraint. In default only the stage one, two and seven are executed to handle the critical Web service.

B. The RepeatAfter policy:

When an exception event is captured, the policy of handling RepeatAfter allows to repeat the execution of a Web service after its execution. A policy of handling exception RepeatAfter (Exception e, WebServiceRepeatWs), when a corresponding event of exception e is captured is defined in Figure 3:

-
- PSEUDO CODE 2.** The RepeatAfter policy
1. CreateSilentPlace (S, simpleTransition, simpleTransition). // Create a silent place S between two simple transitions
 2. CreatePlace (Sra, Seq, simpleTransition, simpleTransition) // Create a new sequential composed place Sra between two simple transitions.
 3. ReplacePlace (RepeatWs, Sra). // Replace the composed place RepeatWs by the place Sra.
 4. RemoveToken (RepeatWs). // A token of state is removed from the place RepeatWs .
 5. AddExceptionToken (S). // Add an exception token to a silent place S.
 6. AddBefor (RepeatWs, S). // Add the place S before the repeated place RepeatWs and run the new H-service-net in order to handle the exception.
-

Figure 3. Pseudocode of the repeatAfter policy.

C. The OtherWs policy

The failure of a Web service Ws requires a search for another Web service OtherWs offering, at least, the same functionalities. The policy of handling OtherWs allows running another alternative Web service in case the Web service fails.

A policy of handling of exception OtherWs (Exception e, WebServiceWs, WebServiceotherWs) when a corresponding event of exception e is captured is defined as in Figure 4:

PSEUDO CODE 3. The OtherWs policy

1. DisableWs (Ws). // Cancel the fail Web service Ws.
2. CreatePlace (otherWs, simple, simpleTransition, simpleTransition). // Create a new place OtherWs between two simple transitions.
3. RemoveToken (Ws). //Remove the token of state in the place Ws
4. AddExceptionToken (otherWs). // Add a token of exception to the place otherWs.
5. CreatePlace (So, seq, simpleTransition, simpleTransition) // Create a new composite place So between two sequential simple transitions
6. ReplacePlace (Ws, So). // Replace the place Ws by the composite place So.
7. AddTo (So, Ws). //Ajouter the service Web to be cancelled in the made up place So.
8. AddAfter (Ws, otherWs) // Add after the place Ws the place otherWs and run the new H-service-net in order to handle the exception.

Figure 4. Pseudocode of the OtherWs policy.

During the execution of the network in Figure 1, an exception of time out is raised from the flight ticket reservation service Ws3. In order to handle the TimeOut exception first, the Web service Ws3 is cancelled, and then a repetition of the same service is performed using the RepeatAfter policy and the network H-Service-Net will become like Figure 5.

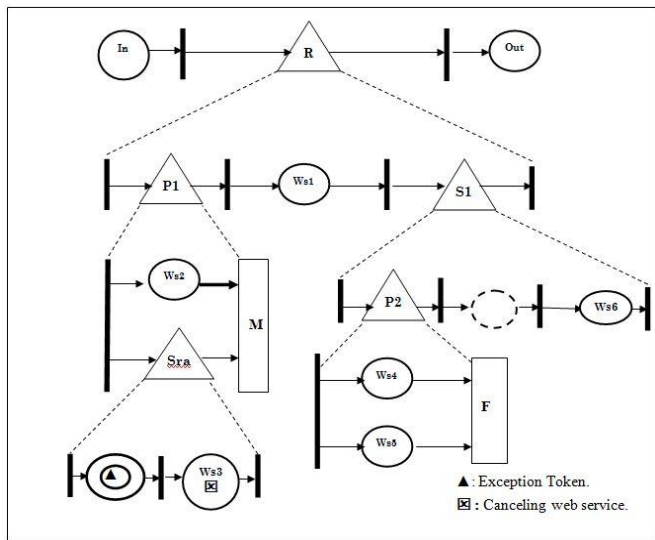


Figure 5. H-Service-Net before the execution of RepeatAfter policy.

After the execution of the RepeatAfter policy if another exception of time out is raised, the TimeOut policy calls another Web service using the OtherWS policy and the H-Service-Net network will become as Figure 6. We note that during the execution of the exception, the token of exception has become a token of state.

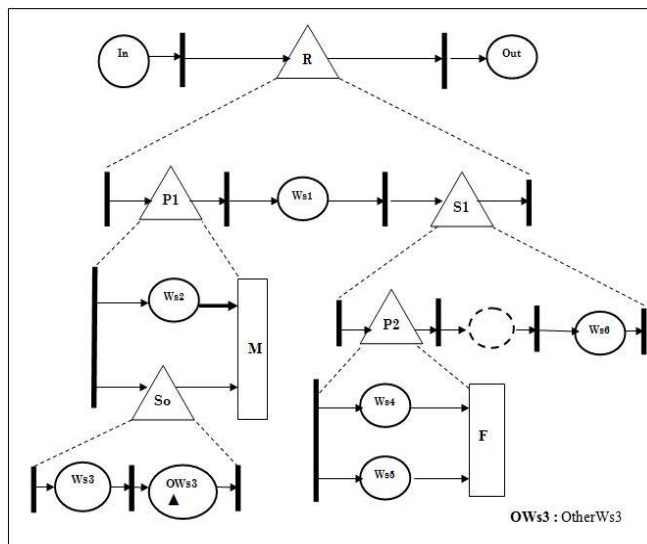


Figure 6. H-Service-Net before the execution of OtherWS

V. IMPLEMENTATION

We developed H-Service-Editor to illustrate the viability of the proposed composition and exception handling techniques presented above. H-Service-Editor is a Web service composition modeling tool with simulation capability; it supports the creation of policies for handling exceptions through the System of handling exception as depicted in Figure 7.

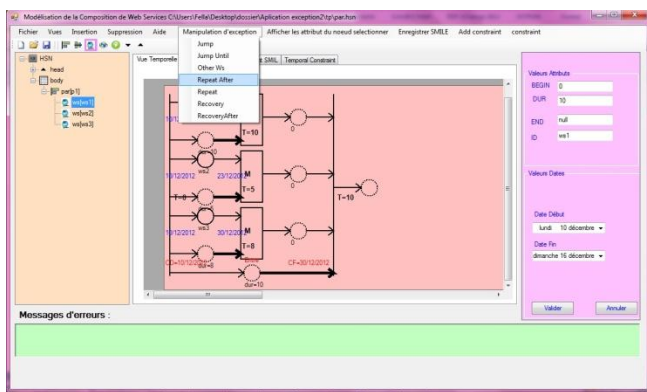


Figure 7. Depicts a screen shot of H-Service-Editor

The authoring environment offers different views that allow the composition of Web services and handling of exception. The different views are:

- The graphical view: it displays the handling of exception for all the policies introduced in this article in the

form of H-Service-Net model. The user can view the different sequences and temporal relations between simple or composite Web services before and after the manipulation.

— XML View: it displays the composition file of the handling of exception in the Extensible Markup Language (XML) standard format document

— The hierarchical view: It allows the representation of services composition in the form of a tree structure. The different services are represented in this area by hierarchical structures similar to the H-Service-Net.

— Error message view: if any temporal conflict is found, the tool displays an error message in order to offer earliest error detection within the editing process of handling exception.

VI. CONCLUSION AND PERSPECTIVE

In order to complete the H-Service-Net model, we have proposed to apply to the H-Service-Net model a manipulation of exception in a hierarchical way. This allows an easy management of time out exception while maintaining the fundamental design of Petri net simple and easy.

Finally, an application (H-Service-Editor tool) of the proposed approach is presented for the modeling of all the policies of handling exception based on our model.

Until now the policies of handling exception are executed manually. The next stage of our research will be to automate the execution of the policies. Further research is needed to consolidate this approach.

REFERENCES

- [1] J. Caoqing, Y. Shi, H. Shanming, X. Hui and Q. Yueming "A Formal Model for Exception Handling in BPEL Process", 2nd International Conference on Computer Science and Network Technology, 2012.
- [2] M. Thirumaran, P.Dhavachelvan, K.Seenuvasan and G.Aranganayagi, "A Novel Approach for Run Time Web Service Exception Handling", Third International Conference on Advanced Computing, 2011.
- [3] Q. Wang, S. Ying, J. Wen, and G. Lv , "Policy-based exception handling for BPEL processes", IEEE International Conference on Information Science and Technology Wuhan, Hubei, China, 2012
- [4] H. Rachid, B. Boualem and M. Brahim, "Self-adapting recovery nets for policy-driven exception handling in business processes" Journal Distributed and Parallel Databases vol. 23 Issue 1, pp. 1 – 44, 2008.
- [5] A. Erradi, P. Maheshwari, and V. Tosic. "Recovery policies for enhancing Web services reliability." In IEEE International Conference on Web Services (ICWS'06), 2006.
- [6] S. Hall "Model-Based Simulation of SOAP Web Services From Temporal Logic Specifications Sylvain", 16th IEEE International Conference on Engineering of Complex Computer Systems, 2011.
- [7] K. Christos, V. Costas, G and Panayiotis1, "Towards Dynamic, Relevance-Driven Exception Resolution in Composite Web Services" University of Athens ,University of Peloponnese, 2006
- [8] K. Lau and C. Tran , "Server-side Exception Handling by Composite Web Services", School of Computer Science, the University of Manchester 2008
- [9] J. Huang, W. Zhu and J. Fu "Automated Exception Handling in Service Composition Using Holistic Planning", IEEE 15th

International Conference on Computational Science and Engineering, 2012.

- [10] R. Prakash, R. Raja "Evaluating Web Service Composition Methods with the Help of a Business Application", International Journal of Engineering Science and Technology Vol. 2(7), pp. 2931-2935, 2010.