QoS Assessment of WS-BPEL Processes through non-Markovian Stochastic Petri Nets

Dario Bruneo, Salvatore Distefano, Francesco Longo and Marco Scarpa

Università degli Studi di Messina, Dipartimento di Matematica
Facoltà di Ingegneria, Contrada di Dio, S. Agata, 98166 Messina, Italy
{dbruneo,sdistefano,flongo,mscarpa}@unime.it

IPDPS'10
24th IEEE International Parallel and Distributed Processing Symposium
Atlanta (Georgia) USA, April 19-23, 2010

Outline

Summary

Service Oriented Architecture

Service Oriented Architecture (SOA) is the most important and effective software paradigm to design Internet-based services.

Fact

Nowadays, Internet-based business processes are shifting from a chaotically organized group of monolithic applications to an ecosystem of services.

In contrast with tightly integrated monolithic applications, the SOA vision of a business process assumes an interaction of loosely coupled reusable Web services (WSs).

Web service orchestration

Using the SOA technology, valued-added WSs can be easily deployed through the invocation and combination of existing WSs.

Fact

Web Services Business Process Execution Language (WS-BPEL) has became the de-facto industrial standard for the composition of WSs as business processes.

Summary

The problem

To allow services to be composed, business relationship between providers and consumers have to be adequately managed:

1. a formal definition of *Quality of Service* (QoS) has to be agreed;
2. effective tools for its measurement have to be developed.

Problem

The QoS of a business process cannot be foreseen at the time the process is written depending on several aspects:

1. availability of the involved services;
2. performance of the involved services;
3. different possible services responses;
4. unexpected services error conditions.

Our technique

We propose a methodology for the evaluation of the performance parameters of a WS-BPEL business process in order to provide QoS-guaranteed composed services at the earliest design phase.

Steps

1. Starting from the WS-BPEL statements and
2. assuming to know the non-functional parameters of the involved services
3. we map the considered WS-BPEL process into a non-Markovian stochastic Petri net (NMSPN)
4. and we numerically solve it by automatic tools
5. in order to evaluate important performance indices such as *service time distributions* and *service availability*.

Summary

An OASIS standard

Definition

WS-BPEL is an OASIS standard XML-based executable language that allow a formal description of the interactions among the partners involved in a business process. The way such interactions take place is through the invocation of Web services.

Version 2.0 provides:

1. management of WSs orchestration;
2. possibility to design a composite service that can be exposed as a new WS;
3. the definition of a model and a grammar for describing the business process logic as a set of activities;

Basic activities

Basic activities implement elementary steps of the process behavior.

Activity	***Description***
<***assign***>	It assigns values to variables.
<***validate***>	It validates the state of variables.
<***wait***>	It waits for the specified amount of time.
<***invoke***>	It synchronously or asynchronously calls a partner WS offered by a service provider
<***receive***>	It waits for a matching message from a business partner.
<***reply***>	It sends a response message in reply to a received message.
<*throw*>	It raises a software fault within a scope. The fault will be caught by the associated fault handler.
<*rethrow*>	It re-throws a fault to the upper scope within a fault handler.
<***empty***>	It does nothing
<***exit***>	It immediately terminates the business process

Fault handlers are defined and executed to manage faults. It is possible to specify one or more fault handlers inner to synchronous <*invoke*> in order to manage WS faults.

Structured activities

Structured activities represent control-flow logic structures. Conditions can be based on the current values of process variables.

Activity	***Description***
<***sequence***>	Collection of activities to be performed sequentially. It terminates when the last activity has completed.
<***if-elseif-else***>	It selects exactly one activity from a set of choices. It completes when the selected activity has completed.
<***while***>	Loop of activities repeated till a condition is true
<***repeatUntil***>	Loop of activities repeated till a condition is false
<***pick***>	It waits for one of several messages to arrive (<*onMessage*>) or for a timeout to expire (<*onAlarm*>) to perform the associated activity.
<***flow***>	Collection of activities to be performed concurrently.
<*foreach*>	K-out-of-n conditional parallel loop
<*scope*>	It defines an execution scope of nested activities
<*compensateScope*>	It starts compensation on an inner scope
<*compensate*>	It starts compensation on all inner scopes

Reference scenario

WS-BPEL programming power introduces a high level of complexity.

Fact

We need to fix a reference scenario in order to make the problem tractable:

1. *a unique <scope> identifying the whole WS-BPEL process is allowed;*
2. *faults are handled within synchronous <invoke> activities only;*
3. *no fault handlers can be associated to the process <scope>;*
4. *<through>, <rethrough>, <compensate> and <compensateScope> activities are not allowed.*

Summary

Petri net

Definition

A *Petri Net* (PN) comprises:

1. a set of places *P* (represented as circles);
2. a set of transitions *T* (represented as bars)
3. a set of input, output and inhibitor arcs

Places may contain tokens.
The marking of a PN is given by the number of tokens in each place.

Transitions

Transitions represent system events.

Definition

A transition t_i is said to be enabled by marking M iff the number of token in each input place is equal or greater than the corresponding input arc multiplicity and if the number of token in each inhibitor place is less than the corresponding inhibitor arc multiplicity.

Any transition t_i enabled by marking M_j can fire, removing as many tokens as the multiplicity of the input arcs of t_i from, and adding as many tokens as the multiplicity of the output arcs to, the corresponding places, obtaining the marking M_k.

non-Markovian stochastic PNs

In NMSPNs, transitions can be:

1. immediate: they are supposed to fire in zero time;
2. timed: their firing time is associated to a generally distributed random variable.

Immediate transitions have higher priorities with respect to timed transitions.

Summary

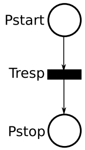
Compositional approach

Our technique is based on a compositional approach:

1. mapping of WS-BPEL constructs and invoked WSs into a set of *basic* NMSPNs;
2. composition of the basic NMSPNs into a *global* NMSPN that models the entire process;
3. analysis of the NMSPN to obtain performance indices of the WS-BPEL process.

WS mapping

We map each WS invoked within the process with the NMSPN:



in which:

1. a token in place `Pstart` models a request to the WS;
2. transition `Tresp` models the WS response time;
3. a token in place `Pstop` models the fact that the WS has finished and has replied.

WS parameters

Each invoked WS has to expose non-functional parameters to properly set the NMSPN attributes:

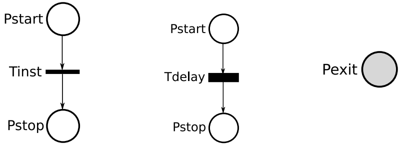
1. response time cumulative distribution function (CDF) to be associated to `Tresp`;
2. information about error conditions in terms of responses probabilities (in case of synchronous WS).

The probabilities associated to WS responses (correct or fault messages) are not used into the WS mapping but in that of synchronous $<$*invoke*$>$.

Summary

Internal activities

Activities that are not related to the interaction with external WSs:



1. activities whose execution time can be neglected (<*assign*>, <*validate*>, <*empty*>): transition `Tinst` represents such negligible amount of time;
2. <*wait*> activity: transition `Tdelay` will present a deterministic distribution to represent the amount of time that the process has to wait;
3. <*exit*> activity: place `Pexit` has a halt semantic associated.

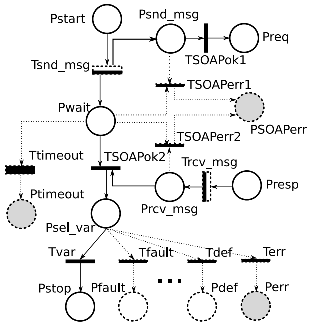
External activities

Activities that are related to the interaction with external WSs:

1. $<$*invoke*$>$;
2. $<$*receive*$>$;
3. $<$*reply*$>$.

We need to distinguish between synchronous and asynchronous $<$*invoke*$>$ activities.

Synchronous <*invoke*> activity (1)



1. The process establishes a SOAP connection with the remote WS;
2. it sends the SOAP message with the request;
3. it waits until a reply is receive or a timeout has fired.

Synchronous <*invoke*> activity (2)

1. transitions `Tsnd_msg` and `Trcv_msg` model the delay related to SOAP interaction (such delay can be neglected or not);
2. transitions `TSOAPerr1`, `TSOAPerr2`, `TSOAPOk1` and `TSOAPOk2` and place `PSOAPerr` model the possibility of a SOAP fault (they are optional);
3. transition `Ttimeout` and place `Ptimeout` model the possibility for a timeout to expire during the SOAP connection (they are optional).

Synchronous <*invoke*> activity (3)

The remote WS can reply:

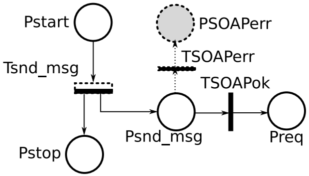
1. with an ordinary output message: transition `Tvar` fires and the WS-BPEL process continuous with the next activity (place `Pstop`);
2. with a fault message. In such case:
   1. the corresponding inner fault handler exists and it is invoked (transition `Tfault`);
   2. the corresponding fault handler does not exist but there is a default fault handler and it is invoked (transition `Tdef`);
   3. no fault handlers can be found and the process exits with an error (transition `Terr` and halt place `Perr`).

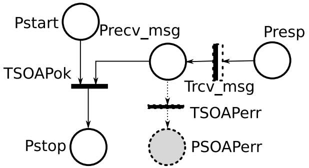
Inner fault handlers activities have to be composed with places `Pfault`, `Pdef` and `Pstop`.

The weights of transitions `Tvar`, `Tfault`, `Tdef` and `Terr` has to be set accordingly to the probabilities exposed by the WS.

Asynchronous <*invoke*>, <*receive*> and <*reply*> activities

They are a subset of the synchronous <*invoke*>.





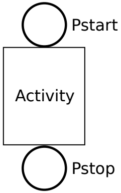
In all such NMSPN places `Preq` and `Presp` need to be composed with places `Pstart` and `Pstop` of the NMSPN mapping of the corresponding WS.

Summary

Semantic

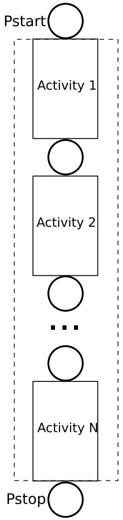
Structured activities:

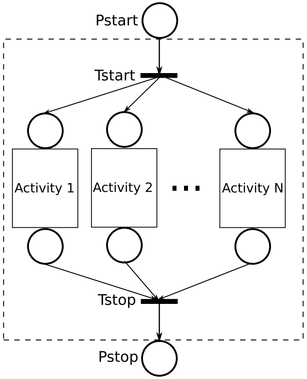
1. determine the order in which a collection of activities is executed;
2. can be nested and combined in arbitrary way;
3. can create complex structures.



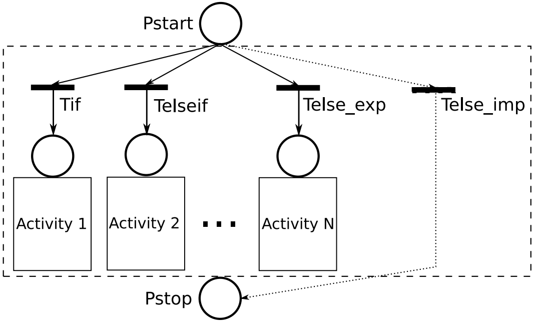
Such NMSPN symbolically indicates either a basic or a structured activity.

<*sequence*> and <*flow*>



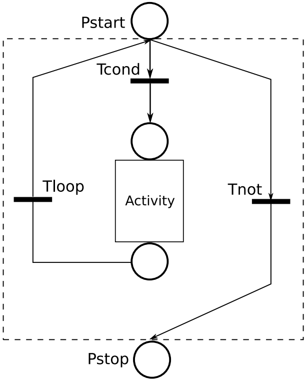


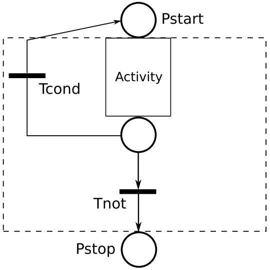
<*if*>



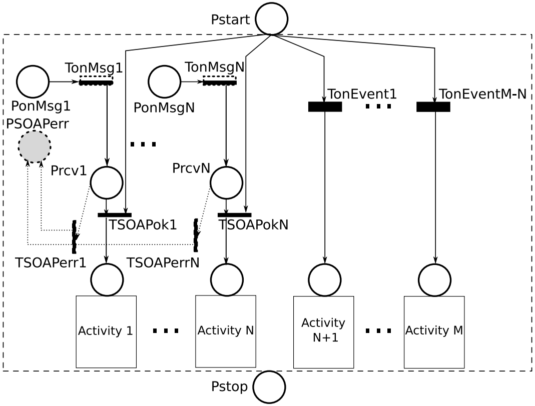
Conditional statements are modeled in a probabilistic manner. The programmer will be able to set the weight in an appropriate way.

<*while*> and <*repeatUntil*>





<*pick*>



1. an <*onMessage*> is equal to a <*receive*>;
2. an <*onAlarm*> is equal to a <*wait*>.

Summary

The model

The stochastic process underlying the *global* NMSPN that represents the WS-BPEL process:

1. presents absorbing states;
2. can be solved to perform a transient analysis to obtain the CDF of the process response time for each possible response;
3. can be solved to perform a steady state analysis to obtain the probabilities of each process response or of each error.

General response time CDF and probability

Let be C_e the set of marking in which there is a token in almost one of the `Pexit` places:

$$\begin{aligned} C_e &= \Big\{ M \in RS | \#(\texttt{Pexit}_0, M) == 1 \bigvee \cdots \\ &\qquad \cdots \bigvee \#(\texttt{Pexit}_{N_e - 1}, M) == 1 \Big\} \end{aligned}$$

then

$$\Theta_e(t) = P[M(t) \in C_e], \; \forall t \geq 0.$$

is the CDF of the response time. It is a defected distribution and

$$\theta_e = \lim_{t \to +\infty} \Theta_e(t)$$

is its steady state probability.

i^{th} response time CDF and probability

Let be C_i the set of marking in which there is a token in place $\texttt{Pexit}_i$:

$$C_i = \{M \in RS | \#(\texttt{Pexit}_i, M) == 1\}$$

then

$$\Theta_i(t) = P[M(t) \in C_i], \ \forall t \geq 0.$$

is the CDF of the response time for response i. It is a defected distribution and

$$\theta_i = \lim_{t \to +\infty} \Theta_i(t)$$

is its steady state probability.

Normalized CDFs and MTTR

Response time distribution given that no errors occurred:

$$F_e(t) = \frac{\Theta_e(t)}{\theta_e}$$

and response time distribution given that response i^{th} has occurred:

$$F_i(t) = \frac{\Theta_i(t)}{\theta_i}$$

Mean time to response for i^{th} response:

$$MTTR_i = \int_0^{+\infty} t \cdot F_i(t)dt$$

Summary

Error places

The model is able to give information about the *environmental* errors:

1. SOAP errors: almost a token in one the `PSOAPerr` places;
2. synchronous <*invoke*> timeout: almost a token in one of the `PTimeout` places;
3. not handled faults: almost a token in one of the `Perr` places;

Error probabilities

The probability to have one of the unexpected errors is:

$$\rho_e = \lim_{t \to +\infty} P[\#(\mathtt{P}_e, M) == 1]$$

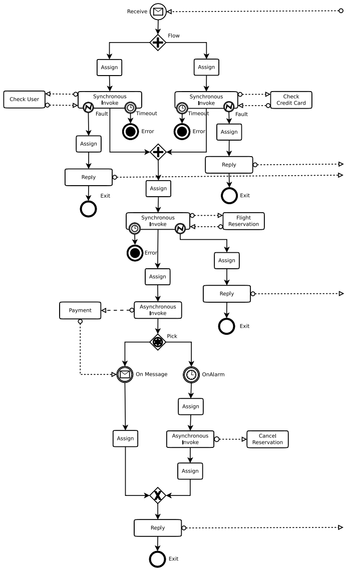
where $\mathtt{P}_e$ is the considered error place.

$$\rho = 1 - \sum_{e \in E} \rho_e$$

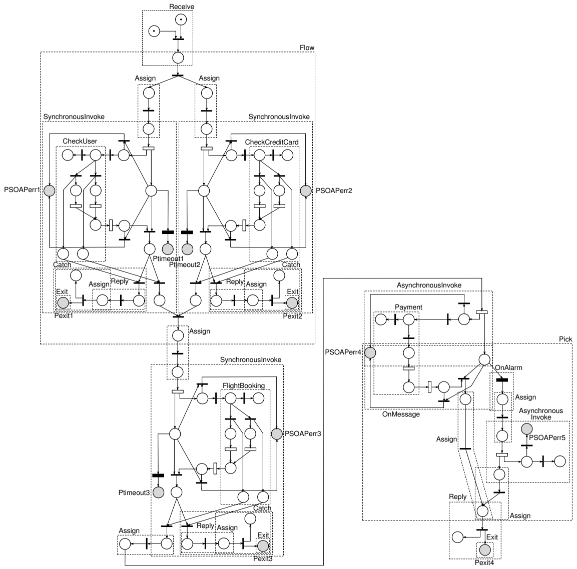
is the probability for the process to complete properly.

Summary

The WS-BPEL process



The corresponding NMSPN



Solution of the model

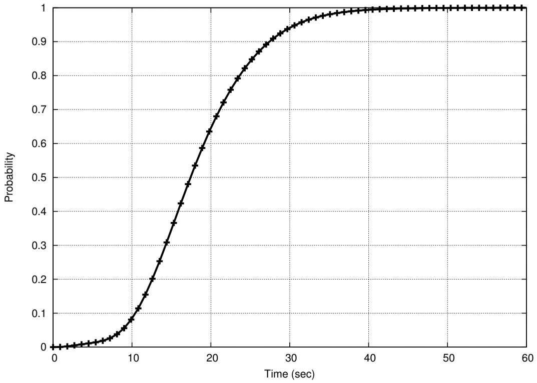
The NMSPN model has been analytically solved by mean of the WebSPN tool.

WS operation	***Type***	$\lambda_{rt}(\text{sec}^{-1})$	p_c	p_f
Check user	synchronous	0.33	0.99	0.01
Check credit card	synchronous	0.33	0.99	0.01
Flight reservation	synchronous	0.25	0.99	0.01
Payment	asynchronous	0.25	-	-

Moreover:

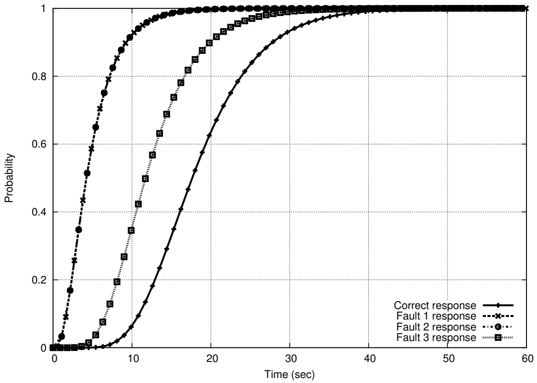
1. SOAP mean delay: $1sec$;
2. SOAP fault probability: 0.01
3. synchronous $<$*invoke*$>$ timeout: $20sec$;
4. $<$*onAlarm*$>$ timeout: $15sec$.

General response CDF and MTTR



MTTR = 18.2*sec*

Different responses CDFs, MTTRs and steady state probabilities



Response	***Probability***	***MTTR (sec)***
Correct response	0.884	18.6497
Fault 1 response	0.009	4.9403
Fault 2 response	0.009	4.9403
Fault 3 response	0.009	12.6497
<*invoke*> timeouts	0.011328	-
SOAP errors	0.075810	-

Summary

Conclusions

1. our analytic technique is able to compute non-functional parameters of WS-BPEL process, starting from those exposed by the invoked WSs;
2. the parameters are evaluated at early design phase before the WS-BPEL process is actually implemented with consequent time and money saving;
3. the non-functional parameter we compute are the same that we need from WSs and so they can be exposed by the WS-BPEL process if it is going to be invoked by another process;
4. performance indices can be used for the management of Service Level Agreements between commercial parties in a SOA contest;
5. our technique can be also used for the on-the-fly selection of the WS to invoke between similar ones when it is necessary to maximize a particular quantity (reliability, response time, etc).