Paper

Model-Based Availability Evaluation of Composed Web Services

Mauro Iacono[1] and Stefano Marrone[2]

[1] *Dipartimento di Scienze Politiche, Seconda Università degli Studi di Napoli, Caserta, Italy*
[2] *Dipartimento di Matematica e Fisica, Seconda Università degli Studi di Napoli, Caserta, Italy*

Abstract—Web services composition is an emerging software development paradigm for the implementation of distributed computing systems, the impact of which is very relevant both in research and industry. When a complex functionality has to be delivered on the Internet, a service integrator can produce added value by delivering more abstract and complex services obtained by composition of existing ones. But while isolated services availability can be improved by tuning and reconfiguring their hosting servers, with Composed Web Services (CWS) basic services must be taken as they are. In this case, it is necessary to evaluate the composition effects. The authors propose a high-level analysis methodology, supported by a tool, based on the transformation of BPEL descriptions of CWS into models based on the fault tree availability evaluation formalism that enables a modeler, unfamiliar with the underlying combinatorial probabilistic mathematics, to evaluate the availability of CWS, given components availability and expected execution behavior.

Keywords—automatic model generation, Composed Web Services, fault tree, model composition, service availability.

1. Introduction

Quantitative evaluation of characteristics of distributed systems is a difficult task, due to the complexity of the systems and to the need for efficient models evaluation. In many fields, classical mathematical evaluation frameworks have been structured into higher level methods that hide the complexity of the mathematical approach by encapsulating it into graphical (or other kinds of) formalisms, easier to manage, with the aim of enabling the use by non-expert users.

Two classical examples of such high level methods are given by Fault Trees and Stochastic Petri Nets. Fault Trees[1] (FT) is a graphical formalism to model the combined probability of a complex event (called fault, since the technique is typical in reliability/availability engineering), given the interrelation of (statistically independent) basic events, that are characterized by a known (fault) probability, specified by a number of intermediate events that result by elementary conjunctions or disjunction of basic events. Such a high-level method allows an expert in system engineering to exploit the benefits of probability computation in large systems, without the need for explicit knowledge in the field and with the additional benefit that, whatever the model complexity, the modeling process is less error-prone than writing the actual equations and scales better. Stochastic Petri Nets (SPN) [1] is a graphical formalism to model the dynamic behavior of concurrent systems that can be evaluated by Markov chains. In SPN the system is modeled by places, characterized by a marking in terms of tokens that represent a part of the system state, and transitions that modify the markings of the places to which they are connected according to exponentially distributed firing rates. Such a method allows a modeler that has a very good knowledge about the system to represent and evaluate it by Markov chains without the need for being familiar with them. Analogously, other methods can be found in literature enabling different ways the use of complex mathematical tools while hiding their details behind a user-friendly framework.

The stratification and the articulation of modern distributed systems, that are built after a system-of-systems logic, pose a second level complexity problem, that consists in modeling such systems by aggregating (sub-)models in order to cope with the explosion of their extension by exploiting the same compositional logic. The availability of higher level methods is not sufficient to enable efficient modeling processes, so additional conceptual tools are needed to supply proper approaches that can scale to more complex and extended system architectures. In this paper we propose a structured approach for the construction of high level probabilistic availability models (based on exponentially distributed variables) for a very common and spread class of distributed systems, such as Service Oriented Architecture (SOA) compliant systems, that are based on the Service Oriented Computing (SOC) paradigm.

The paper is organized as follows. Section 2 presents a brief introduction to FT and their evaluation; Section 3 introduces the SOA and related problems. Section 4 introduces service composition; Section 5 shows the general approach to obtain availability models from BPEL (Business Process Execution Language) documents and is complemented by the description of supporting tool and an example in Section 6 and Section 7. Final considerations and future works complete the paper.

2. Fault Trees

FT is based on the idea that independent undesired events influence the general behavior of a complex system according to its structure. Events effects combinations are repre-

[1] Fault Trees can also be used for a qualitative analysis of the relevance of the role of basic events with respect to the complex event.

sented by two operations, analogous to OR and AND operators of Boolean algebra and logic networks. Two events are combined by an OR operator if each of their single individual contributions is effective on the system, while they are combined by an AND if their contribution is effective only if they happen both at the same time.

Let $A_1 \dots A_n$, $n \in N$ be n events, statistically independent and with probabilities $P_1 = P(A_1) \dots P_n = P(A_n)$, $Pi << 1 \forall i \in 1 \dots n$ (due to the nature of the application field). From a quantitative point of view, the probability P of the event $A = A_i$ OR $A_j, i, j \in 1 \dots n, i \neq j$, represents the probability of the case in which A_i or A_i happens, eventually simultaneously with the other. Such a probability can be computed as $P_i + P_j P_{ij}$, in which $P_i j$ is the probability of having A_i and A_j simultaneously. If A_i and A_j are mutually exclusive, $P_i j = 0$, but generally, being A_i and A_j independent and $P_i, P_i << 1, P_{ij} << P_i + P_j$ and $P \approx P_i + P_j$ (rare events approximation). The probability of the event $A = A_i$ AND $A_j, i, j \in 1 \dots n, i \neq j$, represents the probability of the case in which A_i and A_j happen simultaneously. Such a probability can be computed as $P_i \cdot P_j$, being A_i and A_j independent.

Both operators can be generalized to any number of operands. The reader will find the analysis of the generalized AND and OR operations and a more thorough introduction to FT in [2]. For the goals of this paper, it is sufficient to consider that, in the given hypotheses, by properly using the operators to aggregate together basic (and/or non basic) events to obtain non-basic events it is possible to form the different levels of a FT, until a single event (top event) is described as complex event aggregation. Given a complete FT, the analysis of its structure gives the analytical expression of the probability of the top event that can be easily obtained by mechanical substitutions whatever the complexity of the tree.

The possibility or representing as logical operations the combination of events allows for the application of boolean algebra to fault trees. Given a FT, it can be described in terms of a Boolean equation that can be transformed applying the theorems of Boolean algebra. Such transformation can result into a reduction of the model, or can be exploited, as in the following, to extend the use of FT to include fictitious events that enable for the expression of more complicated conditions. Besides AND and OR, Boolean algebra defines the NOT operator, that is not present in the description given in this section for FT. The interpretation of the NOT operator applied to an event (A, with probability P) results in the definition of the complementary event (A', with probability $1 - P$), that can be used in the FT.

3. Service Oriented Computing

The Service Oriented Computing paradigm provides a methodological foundation for the design of complex distributed applications by integrating existing components, namely services.

A service is a software component capable of interacting with other services with a loose coupling logic to provide a functionality that is performed by it in complete autonomy. A service is accessible independently from its implementation details and is designed to be reused. Services are commonly implemented by Web service technology that allows services to be discovered, integrated and used on the Internet, by running them on a server (that can run simultaneously more Web services).

Services are meant to be integrated, knowing their interfaces, in Business Processes (BP) or Workflows (WF). Among the several techniques available for such integration, BPEL [3] is the most spread solution and took the role of standard language for services orchestration. Orchestration is one of the two main strategies for service integration and is based on the idea that an application consists in a centrally specified BP or WF that operates all services involved and is run by a specific executor.

BPEL is a language for the specification of orchestrations, based on Extensible Markup Language (XML). A BPEL document can be assumed as the formal description of the desired service orchestration. The integration of services by a service integrator can produce a significant added value if the composed, more abstract, complex service is designed to provide a well defined Quality of Service (QoS), eventually better than the one offered by isolated services. Isolated services can be fine-tuned by reconfiguring their hosting servers to increase performances or availability, but in the case of Composed Web Services (CWS) basic services have to be taken as they are. In this case a reasonable measure is to evaluate the effects of the composition. In order to perform such evaluation, a quantitative analysis is needed. In the next section an analysis methodology that allows availability evaluation of CWS by transforming BPEL descriptions into FT models exploiting proper patterns is proposed.

BPEL definition of a CWS intrinsically describes the relations by which the availability of component basic services influences the availability of the composed one. Systematic analysis of BPEL language elements allows the definition of equivalent FT patterns that represent their composition effects. With this premises, it is possible to obtain an evaluation method for the availability of a CWS given components availability and the expected execution CWS behavior. Such an approach offers a decision support tool for integrators.

Literature shows a great interest in formal verification of BPEL programs, mainly oriented to correctness verification. The most spread approaches are based on high-level analysis methods (both for quantitative and qualitative evaluation) to ease the understanding of such systems. An important contribution is provided by van der Aalst's transformations from BPEL to Petri Nets (PN) to perform liveness verification [4]–[6]. Studies of non-functional properties of BPEL WFs [7]–[10] by means of formal models are mainly oriented to performance and security rather than reliability and availability. The use of FT for reliability and avail-

ability models generation is consolidated [11]–[15]. In FT models have been generated from system descriptions while in a UML system model is used for automatic generation of Dynamic FT.
Recent research trends explore the applications of these techniques to the wider topic of the cloud computing. Some scientific works related to this aspect are [16]–[19].

4. Proposed Approach

Following literature general orientation, a high-level modeling approach to the stated availability modeling problem has been chosen, founded onto Fault Trees as a basic high-level probabilistic tool. According to this choice, the low-level mathematical details (related to FT) will not be considered in the following and the work will focus on high-level transformations and compositions.
The main aim of this paper is the definition of a relationship between BPEL language constructs and fault tree patterns. CWS availability must be assured in order to provide a sustainable level of QoS. A failure is an event that occurs when the delivered service deviates from correct service in value or time [20]. In this paper a CWS failing is considered if it does not reply at all to requests (delays are not relevant).
The authors apply model-driven techniques to generate formal models of critical services. In particular model-to-model transformations are applied in order to automate the generation of availability models of CWS. The compositional approach that will be used is supported by the main results of compositional failure analysis [21], [22], according to which system failure models can be constructed from component failure models using a process of composition, For a further introduction to automated safety analysis and reuse the reader can refer to [23].
The main problem in evaluating CWS availability is due to distribution and heterogeneity. Assumed the CWSs as bug-free software components, here we only consider faults due to their distributed nature and the dependence of provided service from requested ones, i.e. hosting hardware, messages delivery, timeout, network faults. Thus, faults mainly come from invoked services that run on remote servers. These faults can be reasonably represented by stochastic models. Assuming that remote services can fail and local ones generally cannot, not every fault occurrence brings CWS to a failure since BPEL can mask faults by means of Fault Handlers (FHs) and offers choice constructs. The effective contributions of faults to CWS availability are determined by the WF business logic, that describes the invocation patterns, and by its workload profile (that gives e.g. the estimated branching probability in a choice or average number of activations in a loop). Note that while for components embedded in a system nature and frequency of their failures are known or at least estimable, components of a CWS are remote services that cannot be examined or stress tested, since they do not belong to the same organization. Anyway, a coarse grain stochastic model of remote service failures can be obtained by logs or tests combined with QoS parameters declared by the providers.
In the following the authors propose FT patterns for some BPEL elements. Since this set is not complete the proposed approach must be considered an ongoing work. FTs have been chosen because of their handiness and because more complex modeling tools (multi-state variables, complex repair mechanisms and dynamic issues) are not needed at the state. For each of these constructs, the authors analyze with a top-down perspective how faults of inner activities propagate to construct failures.
Sequence is the simplest BPEL structured activity, and represents the execution of a temporally ordered sequence of activities, each of which is started only if and when the previous one is completed. A Sequence fails if at least one of the (remote) activities fails. Assuming a Sequence of A and B activities, Fig. 1a depicts the corresponding FT model. In this model A and B are characterized by their unavailability. It is important to stress that A and B are depicted as generic middle events since they can be recursively translated by another pattern. Moreover some of them can also be local activities and can be considered impossible events (unavailability equal to 0). Fig. 1b defines the translation of If construct, that branches the execution into two mutually exclusive activities (one of which can be an empty activity) according to a Boolean condition. As activities in then and else branches are mutually exclusive it can be stated that a failure of the If is possible if A fails when A is activated (events that occurs with probability pTH) or B fails when the else branch is chosen (event that occurs with probability $pELSE = 1 - pTH$). The translation is obtained by the introduction of two virtual events pTH and $pELSE$, that account for these considerations. Figure 1c introduces the FT pattern for Fault handler construct, that executes an activity if the related construct fails. As the whole construct fails if both the handled activity S and the catch process (Catch) fail, the best way to translate it into FT is a subtree with an AND gate between the fault events. Figure 1d depicts behavior of BPEL loops (Foreach, While), that repeat a certain activity a given number of times or until a certain condition is satisfied, from the point of view of unavailability. Assuming a loop of an activity S, if we evaluate (from CWS workload profile estimation) the mean number of times the loop is called, loop fails if at least one of these calls fails[2]. BPEL Flow describes the parallel concurrent execution of a certain number of activities. A Flow of $1 \dots n$ fails in the same way as loops, so Fig. 1e represents its FT pattern. The Link construct needs a deeper discussion. When a link is present in a flow construct, two activities are involved: a link source and a link target. The execution of the target activity depends on the termination condition of the source activity. While source activity failure modes are not affected by this construct, target activity execution may change according to ingoing links status and several conditions that characterize its behavior. Due

[2]This can be written under hypothesis of independence of invoked service.

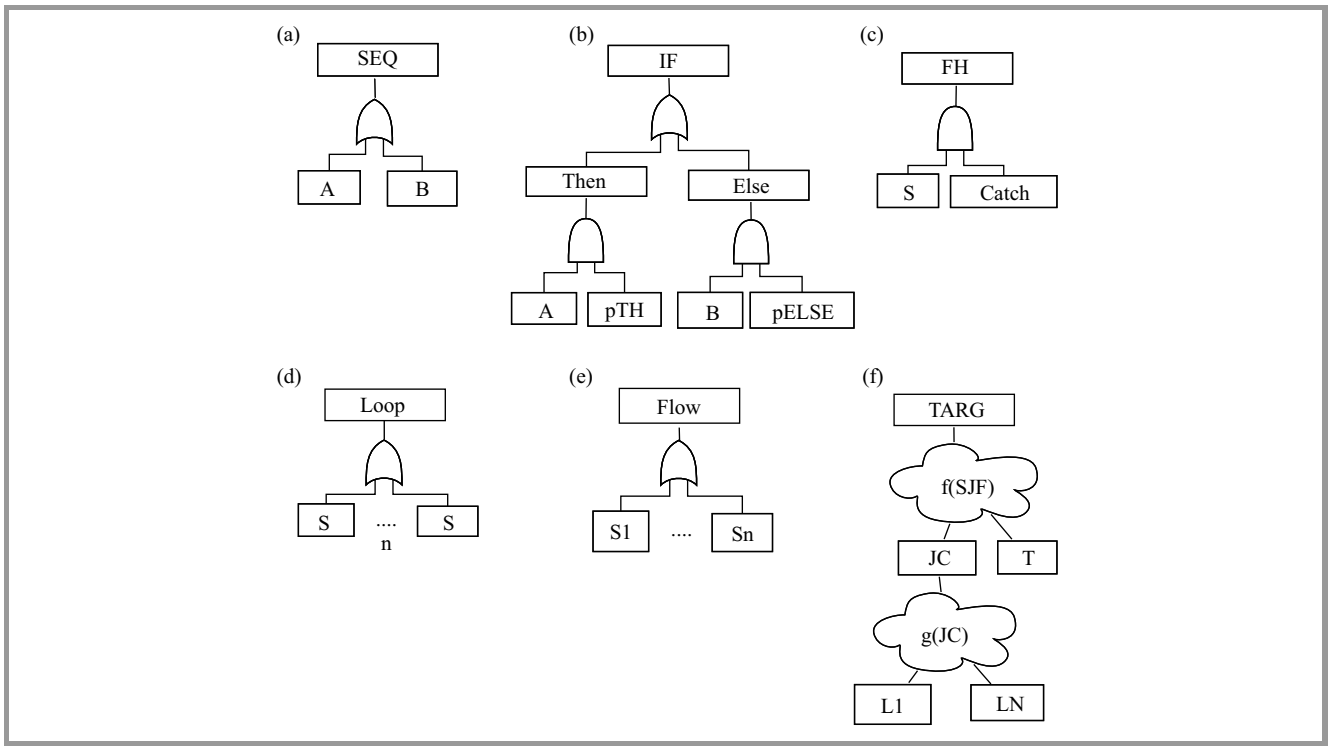

Fig. 1. Fault Tree patterns of relevant BPEL constructs.

to these reasons a different FT pattern must be used. It represents the translation of activity in flows (T) that are target of one or more links (see Fig. 1f). The parameters that influence WF control flow and that directly affect the nature of f and g functions in Fig. 1f, are: *joinCondition* (*JC*) (a logic predicate based on link conditions $1 \dots n$; as this predicate can evaluate to true, target activity is allowed to start) and *suppressJoinFailure* (*SJF*) (when this flag is true, a failure on joinCondition is suppressed, target activity does not start and the fault does not propagate through WF control flow). Otherwise, the target activity may fail on joinCondition failure or on an inner failure.
Such considerations can be summarized into the following equation (also see Table 1):

$$TARG = SJF \cdot T \cdot JC + !SJF \cdot (T + !JC),$$

where ! is the NOT operator, $\cdot$ is the AND operator, $+$ is the OR operator, and where *SJF* is the suppressJoinFailure flag, *JC* is the truth of joinCondition and T is the fault event of target activity.
According to such equation, we can state that:

- if suppressJoinFailure is true, failure of link target can be written as the conjunction of the failure of activity and the success of joinCondition so f is an AND gate while g implements the joinCondition logic predicate,
- if suppressJoinFailure is false, failure of this link target can be written as the disjunction of the failure of activity and the failure of joinCondition so f is an OR gate while g implements the negation of joinCondition logic predicate.

Table 1
Truth table for link target activity

Link Target	SJF	JC	T
0	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
0	1	0	0
1	1	0	1
0	1	1	1

In next sections these patterns are applied to evaluate and compare two example CWS.

5. Support Tools

To support the proposed methodology a tool for the automatic translation of a BPEL file into an analyzable FT has been developed. Moreover the tool takes into account some information that can not be specified into the BPEL file (e.g., probability of then-else branches in if constructs, failure rates of Invoke activities). These data are passed to the tool by a simple properties file. The steps implemented by the tool are depicted in Fig. 2. The tool has been developed in Java and relies on SHARPE [24] for the analysis of generated fault tree.
This is the workflow of the tool in presence of simple services, i.e., where there is one single Web service. In presence of more services, during the phase of properties file

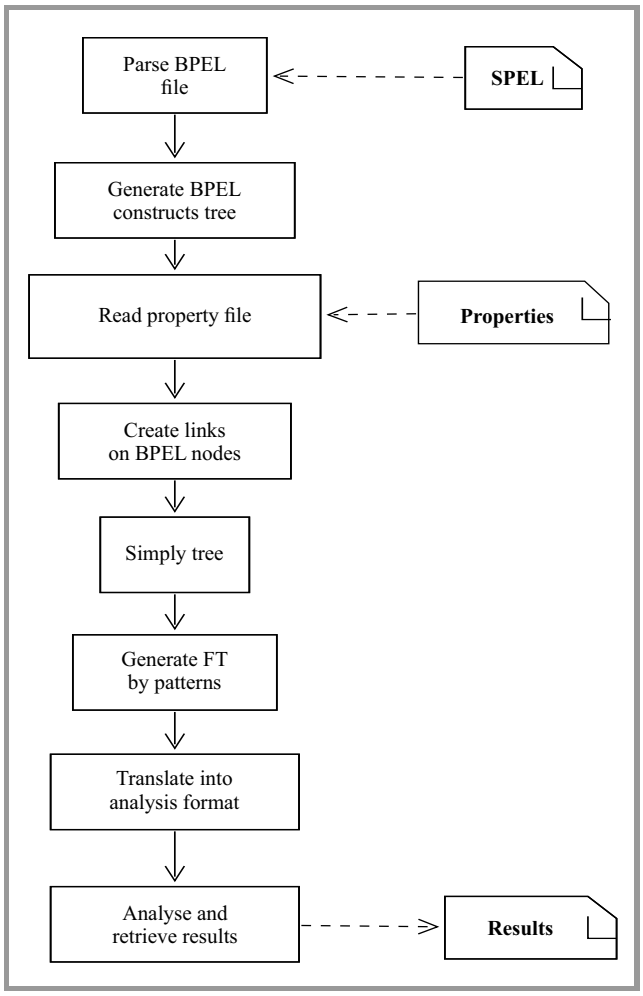

Fig. 2. Tool processing steps.

reading the BPEL constructs tree is explored. Its leaves, that are Invokes constructs, are searched in properties file. In case of an Invoke that is related to another BPEL implemented Web service, the entire analysis process is recursively repeated generating a series of Fault Trees hierarchically organized. Moreover, during the analysis phase, Fault Trees are solved from bottom to up in order to exploit modular analysis of such models.

6. Case Study

In this section proposed FT patterns to the evaluation of availability of a BPEL example WF is applied. The chosen case study is an agri-food information and tracking system. Agri-food information and tracking systems require that information about goods is registered and verified for every item that is produced or transformed in the market. Moreover data vaults are needed in order to store historical data about several aspects of goods, i.e., origin, storage, transportation, composition, processing data. Such registration has to be certified by third parties, eventually authorized and supervised by public authority if requested by the law.

The system to be analyzed requests the registration and logs every request attempt and confirmation on a centralized logging system that is shared with all other dedicated information systems of the same company. The structure of the system is shown in Fig. 3a. The system is composed by a CWS that executes the described orchestration and two subsystems, Log (based on two remote logging services) and Reg (based on three remote registries). Figure 3b describes the high-level message flow performed by the system. The CWS uses the subsystems by requesting a logging operation (Pre with PAck response), the registration (Save with a SAck response) and another logging operation (End with PEnd response).

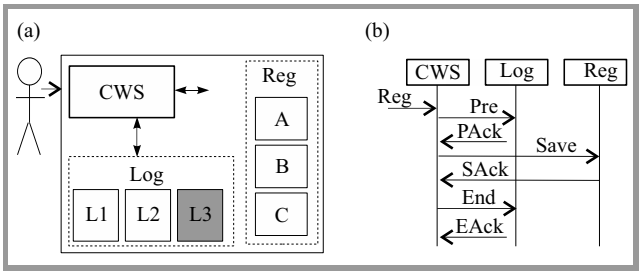

Fig. 3. UML description of case study.

The CWS BPEL implementation can be represented as in Fig. 4.

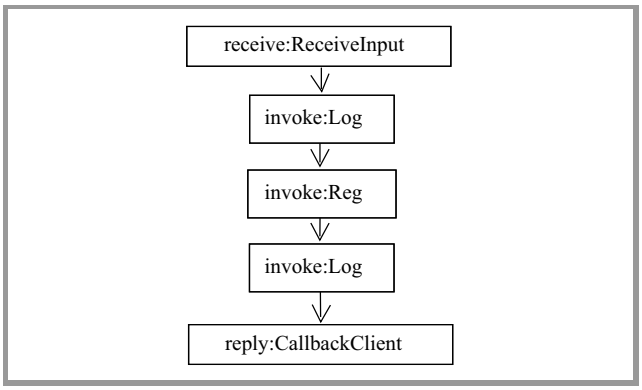

Fig. 4. CWS BPEL implementation.

The Reg subsystem is implemented by another CWS that uses 3 external (geographically distributed) certification services in parallel. Requests to services are cached locally to minimize accesses, and are then performed when the cache is full. Information about the requests that are ready to be sent, cached or processed, are checked and the calculated checksum is logged. Requests arrive in batches of 5 and it is tolerated 1 processing problem per batch.

Processing of batches is implemented by a BPEL Foreach. For each item, the local WS managing the cache is invoked to evaluate if the request can be cached (and just the logging has to be performed) or data has to be sent to registries (in this case, by invoking local services results of registration operations are validated, operations are logged

and the cache is reset). The interaction with the registries is executed by a BPEL flow.

To show the effectiveness of the approach, two versions of the CWS are presented and examined, with slight differences. The two variants are depicted in Fig. 5 (a simple version) and in Fig. 6 (the fault tolerant version).

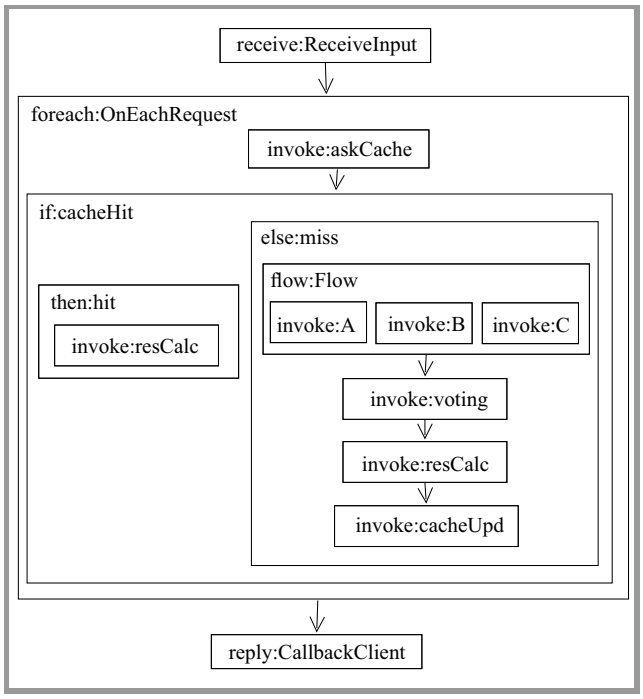

Fig. 5. Simple version of REG service.

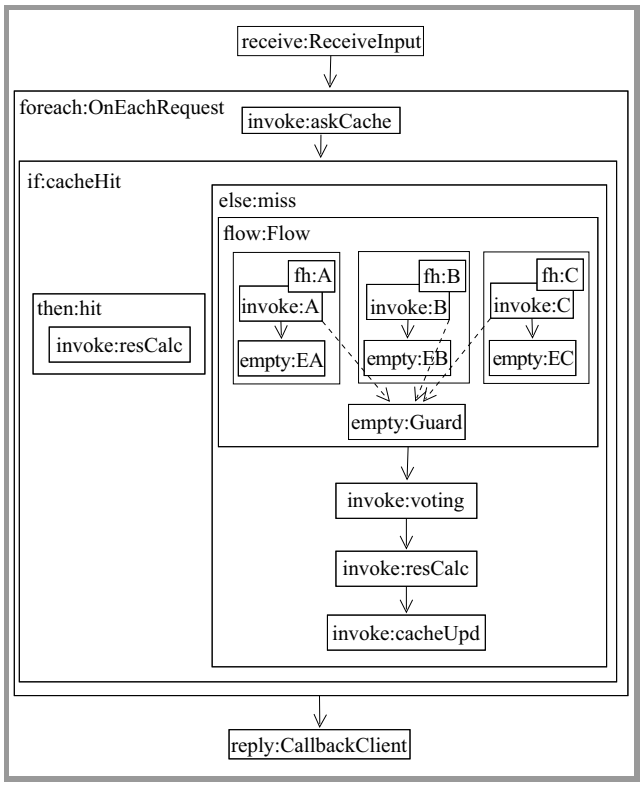

Fig. 6. Fault tolerant version of REG service.

In the first, a fault in one of the service invocations causes a fault of the Flow activity and of the CWS. The second one is designed to complete successfully if at least one of the parallel invocations is successful[3] and join failures are not suppressed[4]. To obtain the same final result, in this implementation failed registrations are externally rescheduled as off-line background operations by the Fault Handlers, if at least one external registry recorded the information so that it becomes public and official.

The second version of the CWS is clearly designed to obtain an improvement in the availability of the system, due to the introduction of fault tolerance in the registration. But how much more available is the second version?

In order to analyze availability of the two implementations, only relevant (fault prone) constructs are to be considered. The resulting FTs differ in the branch that represents the Flow construct, and are represented in Fig. 7. The askCache, resCalc, voting and cacheUpd activities are considered to be fault prone. The three remote invocations (namely A, B and C) are obviously fault prone. The common part of the FT is in Fig. 7a. In the first case, the availability of the Flow depends on the availability of A, B and C as in Fig. 7b. In the second case, by simply applying the patterns Fig. 7c is obtained. Anyway, some considerations are useful to obtain the solution in Fig. 7d, that is the correct solution. At first, Guard is not fault prone and contributes with a null fault probability to the composition, and A, B and C do not give contribution because of the presence of their Fault Handlers. Moreover a Link L_x is true when the related Invoke X is successful, so the probability of the event NOT(L_x is true) is the fault probability of X and $\mathrm{NOT}(\mathrm{OR}(L_a, L_b, L_c)) = \mathrm{AND}(A, B, C)$, that is finally the only contribution of the Flow to the FT.

According to considerations previously made, we assume that: service failure rates are $10^{-7}h^{-1}$ for *askToCache*, *resCalc*, *cacheUpdate* and *CallbackClient*, $10^{-2}h^{-1}$ for A and B, $5 \cdot 10^{-3}h^{-1}$ for C, $10^{-6}h^{-1}$ for voting. The authors are also assuming that the probability of execution of the hit then branch is 0.3. With these parameters, the two variants have a CWS MTTF of 1978000 and 2017400 hours. FT models have been evaluated by the SHARPE tool [24]. The Log subsystem is implemented by a third CWS that uses 2 external (geographically distributed) logging services in parallel. The two services are run by the company, but in a remote data center. They are config-

[3]This is obtained inserting three Empty activities in sequence with the three invocations and a fourth Empty activity, that is the destination of three Links originating in the three Invoke activities, true if the Invoke was successful. This activity is executed if the logical OR of the three Links is true. Another solution without the three Empty activities is possible, but due to ambiguities in the standard it could be not correctly supported by all BPEL interpreters. Here sequences are used to ensure that outgoing Links are set to false in case of faults.

[4]Faults generated by an Invoke activity are masked by a Fault Handler to prevent the propagation to the Flow activity, so that it, and eventually the CWS, will fail only if none of the three parallel Invoke activities is successful. Moreover Links from invoked services and a central consensus activity has been introduced in order to evaluate failures of one or more remote activities.

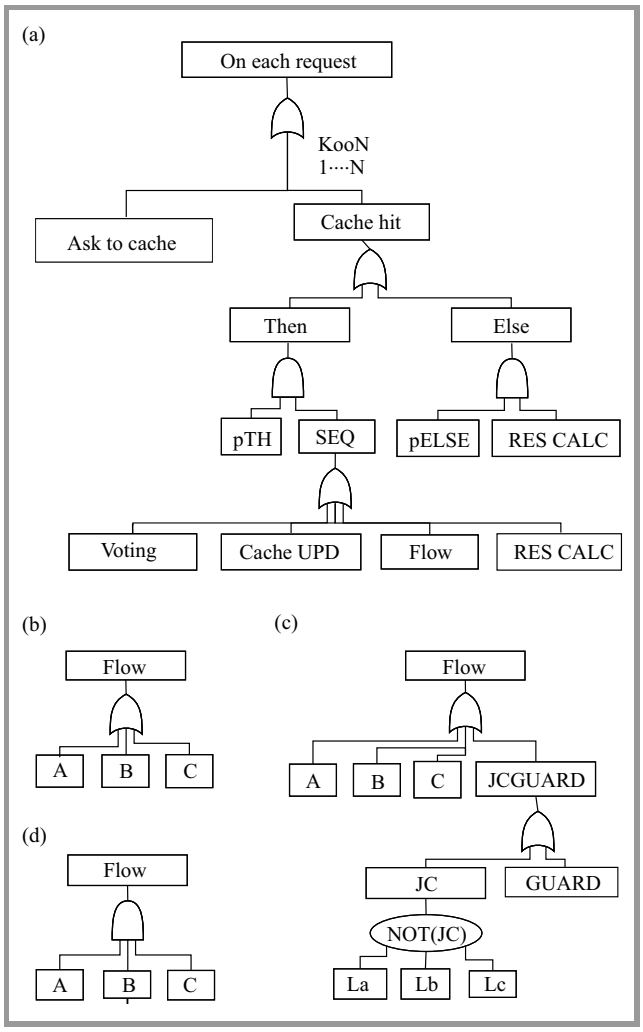

Fig. 7. Generated Fault Trees of CWSs

ured as mirrors and store the same information. Requests to services are implemented by a BPEL flow. Two versions of the CWS are presented: the first requires both the registrations to be successful in order to have a successful completion of the Log CWS (Fig. 8). The second is successful if at least one of the two registrations is successful, as a BPEL Fault Handler instructs the failed mirror to automatically retry the registration according to the other as soon as possible (Fig. 9).

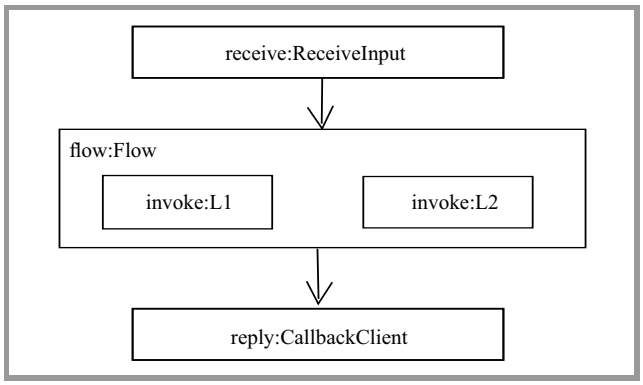

Fig. 8. Simple version of Log service.

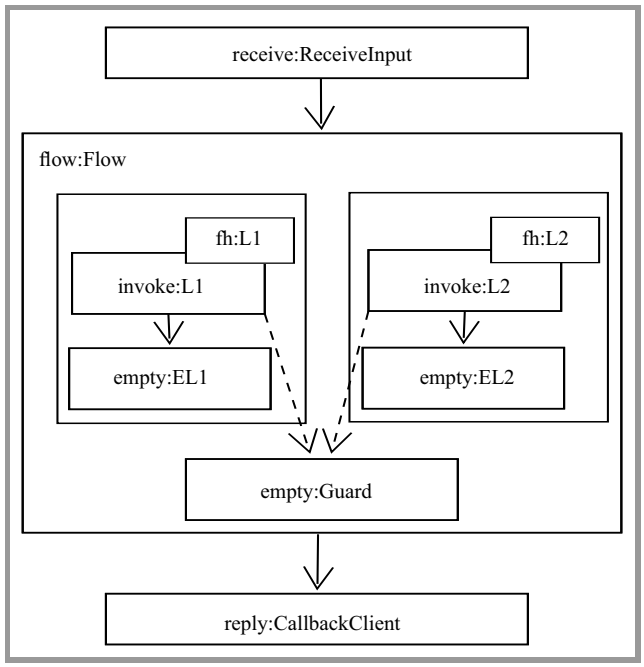

Fig. 9. Fault tolerant version of Log service.

The structure of the two alternative implementations is similar to that of the cases seen for the Reg subsystem, so no further comment will be given here.

Given the compositionality of the approach, the main CWS availability can be obtained using the results of the analyses performed on the other CWS. Four cases can be obtained by combining the two alternative implementations for each subsystem. The general FT for the CWS can be obtained by considering that two subsequent invocations of the Log CWS are completely independent, since they are enclosed in a BPEL Sequence. According to the related pattern, the CWS FT is described in Fig. 10. The complete FTs for the four cases are omitted.

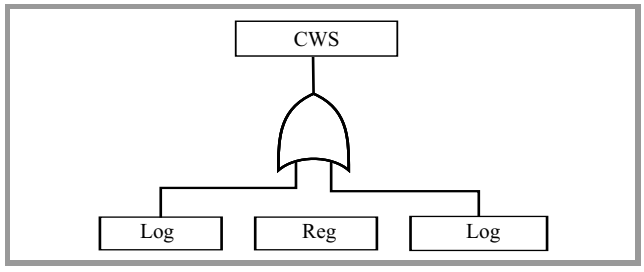

Fig. 10. Fault tolerant version of Log service.

The parameters used in our analysis are summarized in Table 2. Numerical values are failure rates and are expressed in h^{-1}.

According to such parameters, Table 3 describes the overall availability (MTTF) of composed Web service according to the four different configurations of Reg (on rows) and Log (on columns) services.

Table 2
Processes parameters

Parameter	Value
Reg service	
askToCache	10^{-7}
hit	0.8
resCalc	10^{-5}
remoteA	$5 \cdot 10^{-4}$
remoteB	$5 \cdot 10^{-4}$
remoteC	$5 \cdot 10^{-4}$
voting	10^{-9}
cacheUpd	10^{-7}
callbackClient	10^{-8}
Log service	
Log1	10^{-7}
Log2	10^{-7}
callbackClient	10^{-8}

Table 3
MTTF overall analysis

Log Reg	Simple [h]	Fault tolerant [h]
Simple	829530	1241500
Fault tolerant	836400	1256900

One can note that, since the Log process is called twice in the overall process, its influence is stronger so passing from a simple to a complex implementation of this process has a great effect on overall availability. On the other hand the adoption effect of fault tolerant version of Reg process does not give a great advance on overall availability. Such considerations are now not only intuitive and based on qualitative consideration but, by means of the described methodology and tool, can be supported by numerical data.

7. Conclusions and Future Works

In this paper a first step in the translation of a BPEL WF into a formal model in order to evaluate its availability is proposed. For this purpose a FT model is generated having as basic events invoked services whose availability can be measured by black box approaches. The structure of the FT is given by the BPEL WF by applying FT patterns. The effectiveness of proposed approach has been shown by evaluating and comparing, by means of a tool prototype that supports the translation, the availability of two similar BPEL examples, one of which introduces fault masking constructs.

Next steps in this activity will include a further validation of proposed approach by experiments and/or comparisons with models obtained by other formal tools (e.g. Generalized Stochastic Petri Nets). On the other side the authors will study the possibility to include quantitative information, provided at the moment in a separate file, into BPEL, proposing some extension of the language and providing supporting methodologies and tools.

References

[1] G. Balbo, "Introduction to stochastic petri nets", in *European Educational Forum: School on Formal Methods and Performance Analysis*, E. Brinksma, H. Hermanns, and J.-P. Katoen, Eds. *LNCS*, vol. 2090 pp. 84–155. Springer, 2000.

[2] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl, *Fault Tree Handbook*. U.S. Nuclear Regulatory Commission, Washington, DC, 1981.

[3] A. Alves *et al.*, "Web Services Business Process Execution Language Version 2.0 (OASIS Standard)", 2007 [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

[4] W. M. P. van der Aalst, "The application of Petri nets to workflow management", *The J. Circ. Syst. Comp.*, vol. 8, no. 1, pp. 21–66, 1998.

[5] W. M. P. van der Aalst, "Making work flow: On the application of Petri nets to Business Process Management", in *Proc. 23rd Int. Conf. Appl. Theory of Petri Nets*, Adelaide, Australia, 2002. *LNCS*, vol. 2360, pp. 1–22. Springer, 2002.

[6] J. Dehnert and W. M. P. van der Aalst, "Bridging the gap between business models and workflow specifications", *Int. J. Cooper. Inform. Sys.*, vol. 13, no. 3, pp. 289–332, 2004.

[7] J. Xia and C. K. Chang, "Performance-driven service selection using stochastic cpn", in *Proc. IEEE John Vincent Atanasoff Int. Sym. Modern Comput. JVA 2006*, Sofia, Bulgaria, 2006, pp. 99–104.

[8] K. Bhargavan, C. Fournet, and A. D. Gordon, "Verified reference implementations of ws-security protocols", in *Proc. 3rd Int. Worksh. Web Services and Formal Methods WS-FM 2006*, Vienna, Austria, 2006, pp. 88–106.

[9] I. Kim and D. Biswas, "Application of model checking to axml system's security: A case study", in *Proc. 3rd Int. Worksh. Web Services and Formal Methods WS-FM 2006*, Vienna, Austria, 2006, pp. 242–256.

[10] *Web Services and Formal Methods, Third International Workshop, WS-FM 2006 Vienna, Austria, 2006, Proceedings*, M. Bravetti, M. Núñez, and G. Zavattaro, Eds., *LNCS*, vol. 4184. Springer, 2006.

[11] P. Liggesmeyer and M. Rothfelder, "Improving system reliability with automatic fault tree generation", in *Proc. 28th Ann. Int. Symp. Fault-Tolerant Comput. FTCS '98*, Munich, Germany, 1998, p. 90.

[12] J. P. Ganesh and J. B. Dugan, "Automatic synthesis of dynamic fault trees from uml system models", in *Proc. 13th Int. Symp. Softw. Reliab. Engin. ISSRE 2002*, Annapolis, MD, USA, 2002.

[13] A. Bobbio *et al.*, "Comparison of methodologies for the safety and dependability assessment of an industrial programmable logic controller", in *Proc. Eur. Safety and Reliab. Conf. ESREL 2001*, Turin, Italy, 2001.

[14] F. Flammini, N. Mazzocca, M. Iacono, and S. Marrone, "Using repairable fault trees for the evaluation of design choices for critical repairable systems", in *Proc. 9th IEEE Int. Symp. High-Assur. Syst. Engin. HASE 2005*, Heidelberg, Germany, 2005, pp. 163–172.

[15] D. Codetta Raiteri, M. Iacono, G. Franceschinis, and V. Vittorini, "Repairable fault tree for the automatic evaluation of repair policies" in *Proc. Int. Conf. Dependable Syst. Netw. DSN 2004*, Florence, Italy, 2004, pp. 659–668.

[16] L. Wang, "Machine availability monitoring and machining process planning towards cloud manufacturing", *CIRP J. Manufac. Sci. and Technol.*, vol. 6, no. 4, pp. 263–273, 2013.

[17] R. Aoun *et al.*, "Towards an optimized abstracted topology design in cloud environment", *Future Gener. Comp. Syst.*, vol. 29, no. 1, pp. 46–60, 2013.

[18] M. Alrifai, T. Risse, and W. Nejdl, "A hybrid approach for efficient Web service composition with end-to-end QoS constraints", *ACM Trans. on the Web*, vol. 6, no. 2, 2012.

[19] E. Barbierato, M. Iacono, and S. Marrone, "PerfBPEL: A graph-based approach for the performance analysis of BPEL SOA applications", in *Proc. 6th Int. ICST Conf. Perform. Eval. Methodol. and Tools*, Cargese, Corsica, France, 2012, pp. 64–73.

[20] A. Avizienis, J. C. Laprie, and B. Randell, "Fundamental concepts of dependability", Res. Rep. no. 1145, LAAS-CNRS, Apr. 2001.

[21] C. L. Heitmeyer, J. Kirby, B. G. Labaw, M. Archer, and R. Bharadwaj. "Using abstraction and model checking to detect safety violations in requirements specifications", *IEEE Trans. Softw. Eng.*, vol. 24, no. 11, pp. 927–948, 1998.

[22] J. D. Reese and N. G. Leveson, "Software engineering", in *Proc. 19th Int. Conf. Software Engin.*, Boston, MA, USA, 1997, pp. 250–260.

[23] I. Wolforth, M. Walker, Y. Papadopoulos, and L. Grunske, "Capture and reuse of composable failure patterns", *Int. J. Critical Comp.-Based Syst. IJCCBS*, vol. 1, no. 1/2/3, pp. 128–147, 2010.

[24] C. Hirel, R. A. Sahner, X. Zang, and K. S. Trivedi, "Reliability and performability modeling using sharpe 2000", in *Proc. 11th Int. Conf. Comp. Perform. Eval. Model. Tech. Tools TOOLS 2000*, Schaumburg, IL, USA, 2000, pp. 345–349.

Mauro Iacono is a tenured Assistant Professor in Computing Systems. He published around 50 peer reviewed scientific papers and has served as chairman, committee member and referee for many conferences, and as guest editor, editorial board member and referee for several journals. His research is mainly centered on the field of performance modeling of complex computer-based systems, with a special attention for multiformalism modeling techniques, and applications to critical systems and infrastructures, Big Data architectures and dependable systems.
E-mail: mauro.iacono@unina2.it
Dipartimento di Scienze Politiche
Seconda Università degli Studi di Napoli
Viale Ellittico, 31
81100 Caserta, Italy

Stefano Marrone is an Assistant Professor in Computer Engineering at Seconda Università di Napoli, Italy. His interests include the definition of model driven processes for the design and the analysis of transportation control systems, complex communication networks and critical infrastructures. He is currently involved in research projects with both academic and industrial partners.
E-mail: stefano.marrone@unina2.it
Dipartimento di Matematica e Fisica
Seconda Università degli Studi di Napoli
Viale Lincoln, 5
81100 Caserta, Italy