Path-Based Error Propagation Analysis in Composition of Software Services*

Vittorio Cortellessa[1] and Pasqualina Potena[2]

[1] Dipartimento di Informatica
Università di L'Aquila
Via Vetoio 1, 67010 Coppito (AQ), Italy
cortelle@di.univaq.it
[2] Dipartimento di Scienze
Università "G.D'Annunzio"
Viale Pindaro, 42, Pescara, 65127 Italy
potena@sci.unich.it

Abstract. In Service-Oriented Architectures (SOA) composed services provide functionalities with certain non-functional properties that depend on the properties of the basic services. Models that represent dependencies among these properties are necessary to analyze non-functional properties of composed services. In this paper we focus on the reliability of a SOA. Most reliability models for software that is assembled from basic elements (e.g. objects, components or services) assume that the elements are independent, namely they do not take into account the dependencies that may exist between basic elements. We relax this assumption here and propose a reliability model for a SOA that embeds the "error propagation" property. We present a path-based model that generates the possible execution paths within a SOA from a set of scenarios. The reliability of the whole system is then obtained as a combination of the reliability of all generated paths. On the basis of our model, we show on an example that the error propagation analysis may be a key factor for a trustworthy prediction of the reliability of a SOA. Such a reliability model for a SOA may support, during the system development, the allocation of testing effort among services and, at run time, the selection of functionally equivalent services offered by different providers.

1 Introduction

"Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains" [23]. A software system based on the SOA paradigm is developed by assembling software services. Services are offered by providers that hide to users their internal implementation. The users are only aware of a certain behavior that is specified by the service description. Given a set of services $\{s_1; ...; s_n\}$, the functional composition of services can be expressed by the following three operators:

* This work has been partially supported by the PLASTIC project: Providing Lightweight and Adaptable Service Technology for pervasive Information and Communication. EC - 6th Framework Programme. http://www.ist-plastic.org

M. Lumpe and W. Vanderperren (Eds.): SC 2007, LNCS 4829, pp. 97–112, 2007.

$$C ::= (s_1; ...; s_n)|(s_1|...|s_n)|(s_1 + ... + s_n)$$

$(s_1; ...; s_n)$ represents the sequential execution of the services, $(s_1|...|s_n)$ represents the parallel execution and $(s_1 + ... + s_n)$ the possible execution of some services [12]. However, the composition of their non-functional properties, such as the reliability, is not so easy to represent.

In order to keep as simple as possible the modeling aspects of our work, we only consider here the sequential execution of services ([1]). The interactions within this type of composition can be implemented in several ways [23], and we assume that the services communicate by exchanging synchronous messages. In agreement with Parnas [17], sequentially executed services S_i and S_j may undergo two different relations, that are the *Uses* and the *Invokes* relations. The *Uses* relation (here represented as $USES(S_i; S_j)$) means that the service S_i uses S_j for providing its functionality, i.e. as soon as S_i receives a request of service it provides a request of service to S_j. Then, after received an answer from S_j, S_i can elaborate the answer and provide its functionality. The *Invokes* relation (here represented as $INV(S_i; S_j)$) means that the service S_i, at the end of its execution, gives the execution control to S_j. These types of relations will be used in our model for appropriately composing the reliability of services.

Most reliability models for software that is assembled from basic elements (e.g. objects, components or services [9] [12]) assume that the elements are independent, namely the models do not take into account the dependencies that may exist between elements. They assume that the failure of a certain element provokes the failure of the whole system. This assumption is not realistic, for example, in cases of distributed systems where a service interacts with remote services that could run on different operating systems. In this case the services are not independent each other, in fact the middleware that connects them could propagate an error from a service to another one. To some extent, applications that include service wrappers ensuring that a failure is caught in time and close to its source make this assumption more realistic [5].

The independence assumption implies a complete propagation between the services along a path, in the sense that if a service returns an erroneous message then the latter is certainly propagated along the path and the whole system always returns an erroneous message. Goal of this paper is to relax this assumption and consider a wider "error propagation" scenario. We introduce the probability that a service may not produce an erroneous message (i.e. it may "mask" an error to the output, therefore no complete propagation) whether it gets as input value an erroneous message [2].

Our model is based on the composition of scenarios that describe the dynamics of a SOA. Assuming that, for each service provided by the system, we dispose of a scenario represented by a collaboration diagram UML, we provide a technique to generate, for each scenario, all possible execution paths. The reliability of a service is then obtained as the composition of the reliability of all the paths based on the stochastic distribution of the runtime execution of the system, also known as operational profile [16]. The latter consists, on one hand, of the probability that the user executes a certain system service

[1] Being, at the best of our knowledge, the first paper that embeds error propagation in a SOA reliability model, we prefer to focus on this aspect rather than coping with all the above execution operators.

and, on the other hand, of the probability that a service interacts with another one. From the solution of our model, we show that the error propagation analysis may be a key factor for a trustworthy prediction of the reliability of service-based systems.

Information on the SOA may be incomplete. However, if (some) scenarios are not available, then the approach in [22] can be adopted to generate them. Besides, if the operational profile of the system is not (fully) available, then the technique described in [16] can be used to estimate it.

The paper is organized as follows: in section 2 we summarize the reliability estimation models presented in literature and we outline the novelty of our approach; in section 3.1 we introduce a model for SOA reliability that embeds the error propagation factor; in section 4 we provide an example of application of our model, and finally in section 5 we give concluding remarks.

2 Related Work and Novelty of Our Approach

In the last few years many reliability models for software that is assembled from basic elements (e.g. objects, components or services) have been introduced. They can be partitioned in path-based models and state-based models [9]. The former ones represent the architecture of the system as a combination of the possible execution paths, the latter ones as a combination of the possible states of the system. The formulation of the path-based model presented in [21] for component-based systems is the closest one to our approach. From the scenarios of the system, in [21] a *Component Dependency Graph* (CDG) that summarizes all possible execution paths of the system is built.

Inspired by the Yacoub's approach, we introduce in the service domain a model that allows to generate, with a different methodology, the possible execution paths of each scenario representing a service provided by the system. We obtain then the reliability of a service as a composition of the reliability of each generated path. In [21], as in most models for software that is assembled from basic elements (e.g. objects, components or services), it is assumed that the elements are independent, namely the model does not take into account the dependencies that may exist between elements. We relax this assumption here and consider the "error propagation" property in the reliability model for SOA that we propose. This property expresses the probability that a service may propagate an erroneous message when it receives as input an erroneous message. In all the current reliability model this probability is implicitly assumed to be 1, that is complete propagation of errors. In [1] this property has been defined for component-based systems and a formula for its estimation has been provided.

The following aspects characterize the novelty of our approach:

- Our model for SOAs could be adopted to estimate the reliability of software that is assembled from other basic elements (e.g objects or components), but some modifications should be made in order to adapt to other development paradigms. For example, we estimate the reliability of a SOA by partitioning the input domain of the services in equivalence classes. In order to adapt the model to a component-based

system, it is necessary to take into account that a component could offer several services within a scenario. Therefore it is necessary to introduce a new criterion to partition its input domain.
- Our reliability model for SOA is not tied to any particular architectural style or to any particular service-based development process. It can be adopted to obtain a trustworthy prediction of the reliability whether a SOA is completely defined at the design time or certain services are discovered at runtime.
- In our reliability model for SOA we take into account the dependencies that may exist among services. In fact, in order to consider the failures that spread between services, we have embedded the "error propagation" property in the model.
- We model the behavior at runtime of the system by combining the behavior of the system (modelled by a SDG) with the operational profile and the probability of interaction between services. In particular, we consider this probability at the level of input equivalence classes of services.
- Our reliability model is independent from the methodology adopted to represent the scenarios and from the strategy used to generate the possible execution paths. In fact, we assume that each scenario of a service offered by the system is represented by an UML Collaboration Diagram. From the scenario of a service we generate the possible execution paths of the system and we apply to each path the reliability model. However, a scenario could be represented with whatever notation that permits to describe the system scenarios (e.g. the Message Sequence Charts (MSCs) [13] or the UML Sequence Diagrams).

3 Modeling the Reliability of a SOA

In this section we present our approach to build a reliability model of a SOA. We assume that a scenario is available for each functionality that the SOA offers to the users. Each scenario describes the internal dynamics of the functionality, in terms of paths of invoked services. We generate all the possible execution paths, then we estimate the reliability of each path and we obtain the SOA reliability as a composition of its path reliabilities.

At a coarse grain, we can classify the failures that could occur during the execution of a service-based system as follows:

- $crash\ \ failures$, that provoke the crash of the whole system, namely the system straightforwardly stops its execution.
- $no-crash\ \ failures$, that do not provoke the immediate termination of the whole system, but they manifest themselves by returning an erroneous message. This message may either propagate to the system output (thus generating a failure) or it can be masked along its path to the output (thus without actual effects on the system reliability). A finer grain classification of $no-crash\ \ failures$ can be made, but it is out of the scope of this paper.

We focus our attention only on $no-crash\ failures$. Let us assume that, when a service is executed in a path, if it receives as input a correct message then it can fail and

introduce an error in the path with a certain probability (i.e. probability of failure on demand, see section 3.2). Instead, if the service receives as input an erroneous message, then we assume that it may correct the erroneous message that it has received thus masking the internal error to the external outputs.

In the remainder of this section we first describe how to obtain all the execution paths from scenarios, then we introduce our model for the reliability of a single path, and finally we compose those reliabilities to model the whole SOA reliability.

3.1 Generating Execution Paths from Scenarios

Let S be a service oriented architecture composed by n elementary services. Let els_k be the name of the k-th service $(1 \leq k \leq n)$. Let us assume that the input domain of els_k is partitioned in ncl_k disjoint equivalence classes, and that erroneous messages in input that are out of the service domain can be automatically detected and discarded (e.g. with a service harness that filters them), therefore we deal only with erroneous messages that fall within the service domain. The input equivalence classes of each service can be determined in various ways, e.g. using the outlines of the domain testing [11].

Through the composition of its n elementary services, the SOA offers m external services $exts_k$ (i.e. system functionalities) to users, as illustrated in Figure 1.

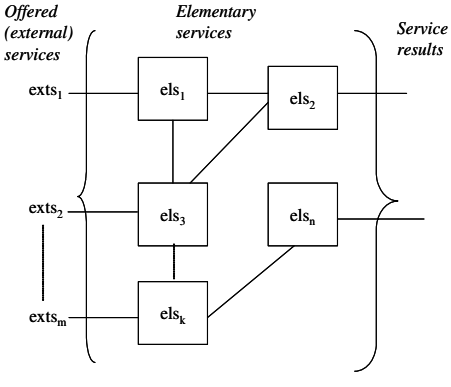

Fig. 1. A schematic representation of elementary and external services in a SOA

Let us assume that for each offered service we dispose of a scenario describing its dynamics (e.g. as an UML Collaboration Diagram) in terms of interactions that take place between elementary services to achieve the goal of the external service ([2]).

Basing on the structure of the Component Dependency Graph (CDG) in [21], we associate a Service Dependency Graph (SDG_k) for each external service $exts_k$, starting from its Collaboration Diagram. SDG_k is a directed graph that describes the behaviour

[2] Note that if the diagrams are incomplete or inconsistent, then the approach in [4] can be adopted to define a reasonably complete and consistent set of diagrams.

of the system (in terms of its possible execution paths) when the external service $exts_k$ is executed.

Another strategy can be used to generate the possible execution paths from each scenario. In [20] Uchitel et. al. synthesize the behaviour of the system from a set of scenarios. This can be done without changing the structure of our reliability model because it gets as input only the possible execution paths.

Definition 1: Service Dependency Graph "SDG_k" - A Service Dependency Graph is defined by $SDG_k = \langle N, E, s, t \rangle$, where:

- $\langle N, E \rangle$ is a directed graph,
- s is the start node, t the termination node,
- N is a set of nodes in the graph,
- E is a set of edges in the graph.

In Figure 2 we show an example of SDG, whose details are given in the following.

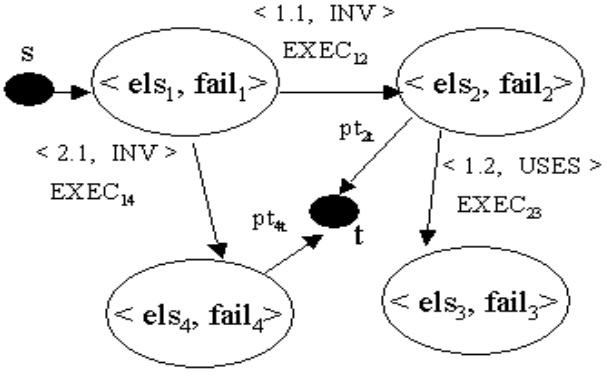

Fig. 2. An example of SDG

Definition 2: Nodes in a SDG - A node i in a SDG represents an elementary service els_i. It is defined by the pair $< els_i, fail_i >$ where $fail_i$ is a vector of ncl_i elements. Each element of $fail_i$, here defined as $fail_i(c)$, $1 \leq c \leq ncl_i$, represents the probability of failure on demand of the service els_i with respect to the c-th equivalence class of its input domain. In other words, $fail_i(c)$ represents the probability that els_i produces an erroneous output given that it has received an input within the equivalence class c.

Definition 3: Directed Edges in a SDG - For each pair of nodes i and j, a directed edge represents the invocation of the service els_j from the service els_i. The invocation is stochastically ruled by the matrix $EXEC_{ij} = [exec_{cd}]$, made of $ncl_i \cdot ncl_j$ elements, where an element $exec_{cd}$ ($1 \leq c \leq ncl_i$, $1 \leq d \leq ncl_j$) represents the probability that the service els_i maps an element of its c-th input equivalence class into an element

of the d-th input class of els_j. In other words, the $EXEC_{ij}$ matrices represent the operational profile of the SOA at the level of input equivalence classes([3]).

Each edge is also labeled with a pair $< p.num, MODE >$. $p.num$ is a composed label, where p identifies the p-th path of execution of the system, and num is a progressive number that determines the sequence of the messages along the p-th path ([4]). $MODE$ is a label that may assume the values "USES" or "INV" if, with respect to the message $p.num$, the service els_i is tied to the service els_j through, respectively, the *Uses* or the *Invokes* relation (see section 1).

3.2 Modeling the Reliability of an External Service

After built the SDG_k for the external service $exts_k$, its reliability on demand ROD_k can be trivially formulated as a function of its probability of failure on demand $POFOD_k$, as follows:

$$ROD_k = 1 - POFOD_k \tag{1}$$

Let I be the event "the input of the service $exts_k$ is correct", and O the event "the output of the service $exts_k$ is erroneous (i.e. the returned result is not the expected one)". Then the probability of failure on demand $POFOD_k$ can be expressed as follows:

$$POFOD_k = P(I \cap O) = P(I)P(O|I) \tag{2}$$

where $P(I)$ is assumed to be equal to 1, because the reliability on demand of a system is always modelled under the hypothesis that the input of the system is correct (namely as defined by its specifications) [2].

Let nep_k be the number of execution paths of the system generated from the execution of the service $exts_k$, $1 \leq k \leq n$. Under our assumptions, the p-th path ($1 \leq p \leq nep_k$) will be made of a pipeline of n_p elementary services $< s_1, ..., s_{n_p} >$. Following the previous notation, ncl_j represents the number of equivalence classes of the j-th service in the p-th path ($1 \leq j \leq n_p$).

Then $P(I \cap O)$ for the k-th external service can be formulated as follows:

$$P(I \cap O) = P(O|I) = \sum_{p=1}^{nep_k} \left(\sum_{c=1}^{ncl_1} P(O|I_c) \right) \tag{3}$$

where I_c is the event "the input of the service $exts_k$ belongs to the c-th equivalence class of the service s_1 of the pipeline of services of the p-th path". This formula holds under the assumption that the equivalence classes are disjoint (see section 3.1).

[3] Note that input classes have no meaning for the end point t, thus in Figure 2 scalar probabilities pt_{it} label the transitions from each els_i service to t. Analogous scalar probabilities could label transitions from the start node s in case an SDG represents multiple paths with different initial nodes.

[4] Recall that we consider only the sequential composition of the services (see section 1).

The probability $P(O|I_c)$ that the output of the service $exts_k$ is erroneous, given that the input of the service $exts_k$ belongs to the c-th equivalence class of the service s_1 of the pipeline of services of the p-th path, can be reformulated by summing over $1 \leq cn \leq ncl_n$ the probabilities of the events "the last service of the pipeline of services of the p-th path produces an error given that the input of the last service of the pipeline of services of the p-th path belongs to its cn-th equivalence class and that the input of the service $exts_k$ belongs to the its c-th equivalence class". Then we have:

$$P(O|I_c) = \sum_{cn=1}^{ncl_n} P(E_n|I_{cn} \cap I_c) \tag{4}$$

where I_{cn} is the event "the input of the service s_{n_p} belongs to its cn-th equivalence class", E_n is the event "the service s_{n_p} produces an error".

In general, for the j-th service of the pipeline of services in the path we can write the following expression:

$$P(E_j) = P(CI_j) * P(F_j) + (1 - P(CI_j)) * P(NM_j) \tag{5}$$

where CI_j is the event "the input of the service s_j is correct", F_j is the event "the service s_j fails and returns an erroneous result", and NM_j is the event "the service s_j does not mask an error".

For two adjacent services i and j in a pipeline of services (where i precedes j) we can write the following formula based on (5):

$$P(E_j|I_{cj}) = \sum_{ci=1}^{ncl_i} P(T_{cicj}) * [(P(CI_j|I_{cj} \cap I_{ci}) * P(F_j|I_{cj}) + \\ +(1 - (P(CI_j|I_{cj} \cap I_{ci})) * P(NM_j|I_{cj})] \tag{6}$$

We can separately obtain each term of the right-side of (6) as follows:

- $P(T_{cicj})$ represents the probability of the event T_{cicj} "the service s_i maps an element of its ci-th equivalence class to an element of the cj-th equivalence class of s_j"(see section 3.1).
- $P(CI_j|I_{cj} \cap I_{ci}) = (1 - P(E_i|I_{ci}))$,
  $P(CI_j|I_{cj} \cap I_{ci})$ can be recursively estimated. The probability that the service s_j receives a correct input depends on the probability that the services that precede it in the pipeline ($< s_1, ..., s_{j-1} >$) have not produced an error.

  $P(E_i|I_{ci})$ that represents the probability of the event "the service s_i produces an error given that the input of s_i belongs to its ci-th equivalence class" should be estimated by supposing that the input of the service s_j belongs to a certain equivalence class cj of its input domain. In order to keep our model as simple as possible, we assume that the service s_i, given an input in one of its equivalence classes ci, has the same probability to produce an error for each equivalence class of s_j.

- $P(F_j|I_{cj}) = fail_j(cj)$,
  where $fail_j(cj)$ is the probability of failure on demand of service s_j with respect to the cj-th equivalence class of its input domain (see section 3.1). $P(F_j|I_{cj})$ should be estimated by supposing that the input of the service s_i belongs to a certain equivalence class ci of its input domain. In order to keep our model as simple as possible, we assume that the service s_j, with respect to an its equivalence class cj, has the same probability to produce an error for each equivalence class of s_i.
- $P(NM_j|I_{cj}) = P(s_j[x] \neq s_j[x^{'}]|x \neq x^{'} \ \cap \ x, x^{'} belong \ to \ the \ cj-th \ equivalence \ class)$,
  $P(NM_j|I_{cj})$ should be estimated by supposing that the input of the service s_i belongs to a certain equivalence class ci of its input domain. In order to keep our model as simple as possible, we assume that the service s_j, with respect to an its equivalence class cj, has the same probability to not mask an error for each equivalence class of s_i.

Upon estimating the formula (6) for each equivalence class of the last service of the pipeline of services of a path, we substitute this estimation in formula (4). In turn, by back substituting in formulas (3), (2) and (1), we obtain an expression for ROD_k. Summarizing, the input parameters of ROD_k are:

- Transition Probability $P(T_{cicj})$;
- Probability of failure on demand $fail_j(cj)$;
- Probability that the service does not mask an error with respect to one of its equivalence classes $P(NM_j \cap I_{cj})$.

These parameters may be characterized by a not negligible uncertainty. The propagation of this uncertainty should be analyzed, but it is outside the scope of this paper. Several methods to perform this type of analysis can be found, e.g. it has been done in [8] for a reliability model. However, in Appendix we discuss how to estimate these parameters.

In order to provide an operational support to the model that we have introduced here, we have plugged the previous formulas into an algorithm that estimates the reliability on demand ROD of the p-th path of execution of the service $exts_k$ using the SDG_k graph. The *RelEval* algorithm is illustrated below.

We assume that the first node of the path models the service els_1 that is tied trough the *Invokes* relation with the service els_k, $1 < k \leq n$. $PEvet_{c1}$ is a global variable that we use to store, for each service that we find in the graph and that belongs to the pipeline of the path, the probability that it produces an error given that the input of the system (i.e. that one of the service els_1) belongs to its $c1$-th equivalence class. For each equivalence class of the j-th service of the pipeline of services, we evaluate the formula (6) ($\forall c1 = 1, ..., ncl_1$) and we store the results in $PEvet_{c1}$. $MODE$ is the edge label that specifies *Invokes* or *Uses* relations (see section 1). For sake of simplicity, and without loosing generality, we assume that in a path two adjacent edges with the *Uses* relation cannot be found and that the last node of the path is tied to the path with the *Uses* relation. $POFOD_p$ is a structure that we use to store the probability of failure on demand of the p-th path.

$\underline{RelEval\ \ Algorithm}$
$Parameters$
$consumes\ \ the\ \ p-th\ \ path\ \ of\ \ SDG_k$
$produces\ \ POFOD_p$
$Algorithm$
$push\ \ (< els_1, fail_1 >,$
$\forall c1 = 1, ..., ncl_1\ \ determine\ \ PEvet_{c1}\ \ using\ \ (6))$
$while\ \ Stack\ \ not\ \ EMPTY\ \ do$
$pop\ \ (< els_i, fail_i >)$
$if\ \ els_i = \ \ t\ \ (terminating\ \ node)$
$\forall c1 = 1, ..., ncl_1\ \ determine\ \ PEvet_{c1}$
$POFOD_p\ \ = \sum_{c1=1}^{ncl1} \sum_{ci=1}^{ncli} PEvet_{c1}[ci]$
$else$
$push\ \ (< els_j, fail_j > | els_j\ \ is\ \ the\ \ \ service\ \ represented\ \ in$
$the\ \ successive\ \ node\ \ of\ \ the\ \ path)$
$if\ \ (MODE \equiv USES)$
$\forall c1 = 1, ..., ncl_1\ \ determine\ \ PEvet_{c1}$
$using\ \ (6)\ \ with\ \ respect\ \ to\ \ els_j$
$\forall c1 = 1, ..., ncl_1\ \ determine\ \ PEvet_{c1}$
$using\ \ (6)\ \ with\ \ respect\ \ to\ \ els_i$
$end\ \ while$

On the loop problem. In the SGD_k graph there can exist some loop. This is a frequent problem of the path-based reliability models. The problem can be solved either by simplifying the number of paths with the ones observed experimentally during the testing [9], or by introducing the average time of execution of the system and of each service [21]. In the latter case the termination of a path is determined if its average execution time (obtained by summing up the average execution time of each component found along the path) is larger than the average execution time of the system. The average execution time of each service and the average execution time of the system can be estimated with monitoring techniques [3]. The *RelEval* algorithm can be easily modified to embed this termination criterion.

3.3 Modeling the System Reliability

It is easy to understand that the reliability of the whole system ROD can be modeled as follows:

$$ROD = \sum_{k=1}^{m} ROD_k \tag{7}$$

where we recall that ROD_k represents the reliability of the k-th service offered by the system

4 An Application Example

In order to show the practical usage of our reliability model, and the relevance of error propagation, in this section we apply it to an example. We have considered the "bank account example" used by McGovern et al. in [15], thus readers interested to the application details, that we do not provide here, can refer to [15]. We have taken into account two scenarios of the system, illustrated respectively in Figures 3 and 4. Each scenario models the dynamics of an external service provided from the system. After estimated the reliability of both services, the reliability of the whole system has been obtained as their algebraic mean, under the hypothesis of uniform probability for their invocation.

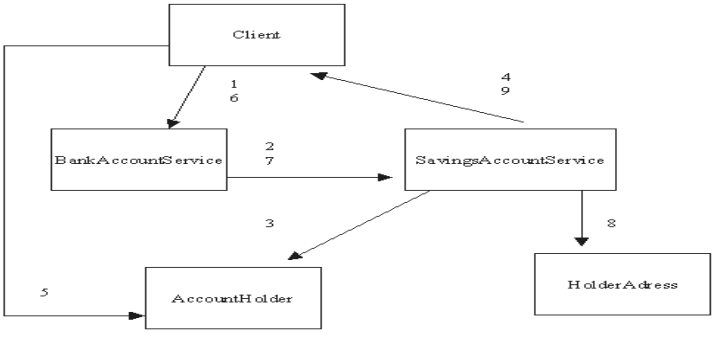

Fig. 3. First scenario of the bank account example

We have conducted two experiments that differ for probabilities of failures and error propagation probabilities of services. We describe the experiments in the following subsections.

4.1 First Configuration: Varying All Error Propagation Probabilities

We have observed the probability of failure on demand of the system while varying, at the same rate, the probability that all services do not mask an erroneous message $P(NM_j)$ (i.e. a measure of the error propagation property), for different values of the probability of failure on demand $fail_j$ of the service $BankAccountService$. So, we have assumed that only one service can introduce an error. The error may or may not be masked by the services in the pipeline of the path that follow the erroneous service, and this depends on their probability of error propagation.

In Figure 5 we report the results obtained in this configuration. Each curve represents the probability of failure on demand of the system while varying from 0.1 to 1 the error

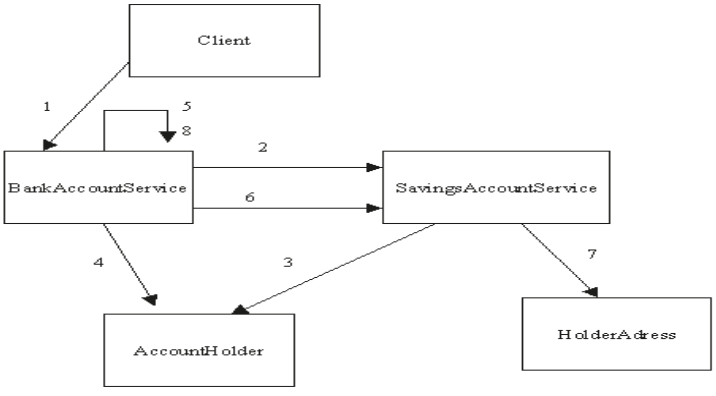

Fig. 4. Second scenario of the bank account example

propagation probability of all services. Curves differ because a different fixed value of the probability of failure of the service $BankAccountService$ has been assigned for each curve. We have obtained the curves by varying this last value from 0.1 to 0.9.

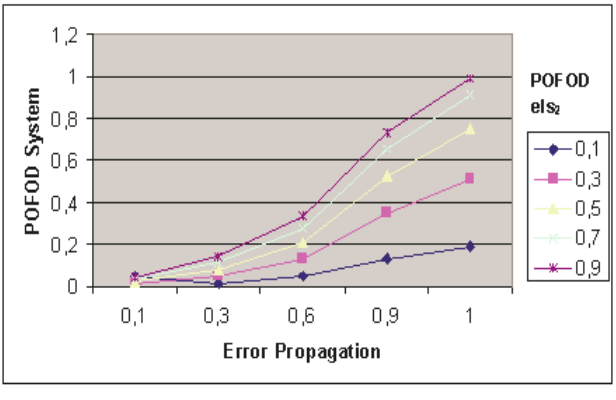

Fig. 5. Model solutions

As expected, for a given value of the probability of failure on demand of the service $BankAccountService$ (i.e. for a given curve), the probability of failure on demand of the system increases while increasing the error propagation of each service of the path (i.e. the probability that an erroneous message produced by the service $BankAccountService$ is not masked by other services that follows it in the pipeline).

This provides a first evidence of the relevance of the error propagation property in a SOA reliability model. In fact, the assumption of independence between failures of services, like in the model presented in [21], corresponds in Figure 5 to the points

of the curves where the error propagation on the x-axis equals 1. They are, for each curve, the maximum values of the system $POFOD$. This means, as expected, that the independence between failures brings to an overpessimistic prediction of the probability of failure on demand of the system. This result confirms that the error propagation analysis is a key factor for a trustworthy prediction of the reliability of service-based systems, and its estimation leads in our model a more precise (and less pessimistic) estimation of the SOA reliability.

On the other hand, for the same value of error propagation probability of the services, the probability of failure on demand of the system decreases while decreasing the probability of failure of $BankAccountService$. This can be observed by fixing a value on the x-axis and observing the values on the curves while growing $POFOD$ of els_2.

4.2 Second Experiment: Varying One Error Propagation Probability

We have observed the probability of failure on demand of the system while varying the probability that the service $HolderAdress$ does not mask an erroneous message $P(NM_j)$, and varying the probability of failure on demand $fail_j$ of the service $BankAccountService$. So, we assume that only one service can introduce an error in the path, and that only one service can correct the error. Furthermore, we have partitioned the domain of the $HolderAdress$ service in two equivalence classes, and we have assumed that the first class is more used than the second one and that the following relation ties the two classes:

$$P(NM_j|I_2) = 1 - P(NM_j|I_1) \tag{8}$$

In Figure 6 we report the results. Each curve represents the probability of failure on demand of the system while varying the error propagation of the first equivalence class of the domain of $HolderAdress$, and with the value of the probability of failure of the $BankAccountService$ service fixed. We have obtained the curves by varying this last

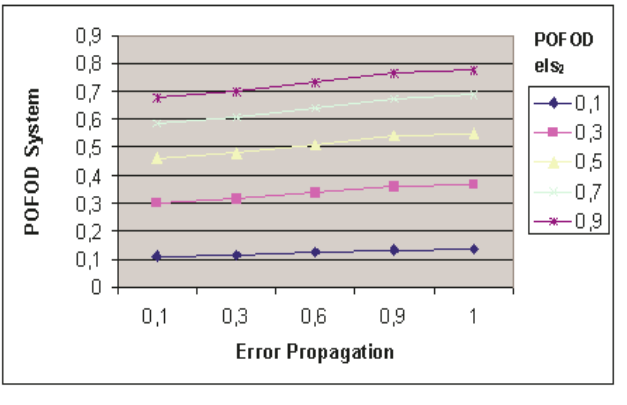

Fig. 6. Model solutions

one from 0.1 to 0.9 and the error propagation of the first class of equivalence of the service $HolderAdress$ from 0.1 to 1.

The partition of the input domain of $HolderAdress$ in two equivalence classes allows to obtain a better estimation of the probability of failure on demand of the system. In fact, basing on (8), for higher values of the error propagation probability of the first class we have lower probability of the error propagation of the second class that it could not be evidenced without domain partition.

A relevant observation is that, for corresponding values of curves in the two experiments, the probability of system failure is higher in the second experiment than in the first one. This is because in the first experiment we assume that all services in the path have the ability to correct an error, whereas in the second experiment we consider an error marking ability only for the service $HolderAdress$ with all the other services always propagating errors.

5 Concluding Remarks and Future Work

In this paper we have introduced a model for the estimation of the reliability of a SOA, based on the reliability of each service and the operational profile, that embeds the error propagation property. The first results that we have obtained supports our intuition that the error propagation may be a key factor for a trustworthy estimation of a SOA reliability.

Our approach can be used at development time to appropriately allocate testing effort. For example, if the SOA reliability is too low, then several alternatives can be easily evaluated to study the sensitivity of the reliability to the SOA modifications, such as replacing a service with a more reliable one. Our approach can be also used at runtime, for example, as a basis for Service Level Agreement negotiation process. Providers may use this model to estimate the Quality of Service that they can provide, given the current status of the system.

The problem of service selection on the basis of their reliability has been widely investigated in the last few year, and it is not easy to solve. In fact, in [7] the authors demonstrate that the problem of service allocation for a composite Web service (i.e. a possible implementation of SOA [24]) is NP-complete.

As future work, we intend to develop the following major aspects of our approach:

- Widening the model experimentation on real world case studies.
- Embedding in our model other specific characteristics of the SOA domain, such as service discovery and run-time service composition.
- Enhancing our reliability model (and the estimation algorithm) by considering the other operators that express the composition of the services, such as parallel operators (see section 1).
- Introducing in our reliability model the probability of failure of a transition between services, that is modelled by an arc in an SDG graph.
- Introducing in our reliability model the *crash failures* (see section 3).
- Embedding our reliability model into a decision support automated framework.

References

1. Abdelmoez, W., et al.: Error Propagation in Software Architectures. In: Proc. of METRICS (2004)
2. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. on Dependable and Secure Computing 1(1) (January-March 2004)
3. Baresi, L., Ghezzi, C., Guinea, S.: Smart Monitors for Composed Services. In: Proc. of the 2nd International Conference on Service Oriented Computing, pp. 193–202 (November 2004)
4. Bertolino, A., Marchetti, E., Muccini, H.: Introducing a Reasonably Complete and Coherent Approach for Model-based Testing. In: Garavel, H., Hatcliff, J. (eds.) ETAPS 2003 and TACAS 2003. LNCS, vol. 2619, Springer, Heidelberg (2003)
5. Cortellessa, V., Singh, H., Cukic, B.: Early reliability assessment of UML based software models. In: Proc. of WOSP 2002, Rome, Italy, July 24-26, 2002 (2002)
6. Diaconescu, A., Murphy, J.: Quality of Service in Wide Area Distributed Systems. In: Proc. of Information Technology and Telecommunications, Waterford, Ireland, pp. 39–47 (October 2002)
7. Esmaeilsabzali, S., Larson, K.: Service Allocation for Composite Web Services Based on Quality Attributes. In: SoS4CO. the First IEEE International Workshop on Service Oriented Solutions for Cooperative Organizations, pp. 71–79. IEEE Computer Society Press, Los Alamitos (2005)
8. Goseva-Popstojanova, K., Kamavaram, S.: Uncertainty Analysis of Software Reliability Based on Method of Moments, FastAbstract ISSRE (2002)
9. Goseva-Popstojanova, K., Trivedi, K.S.: Architecture based-approach to reliability assessment of software systems. Performance Evaluation 45, 179–204 (2001)
10. Hamlet, D., Mason, D., Woit, D.: Theory of Software Reliability Based on Components. In: ICSE 2001 (2001)
11. Kaner, C.: Teaching Domain Testing: A Status Report. In: CSEET 2004. Proc. of 17th Conference on Software Engineering Education and Training (2004)
12. Kokash, N.: A Service Selection Model to Improve Composition Reliability. In: ECAI 2006. Proc. of International Workshop on AI for Service Composition, in conjunction, Riva del Garda, Italy (August 2006)
13. ITU. ITU-T Recommendation Z.120 Message Sequence Charts (MSC99). Technical report, ITU Telecommunication Standardization Sector (1996)
14. Li, J., et al.: An Empirical Study of Variations in COTS-based Software Development Processes in Norwegian IT Industry. In: Proc. of METRICS 2004 (2004)
15. McGovern, J., Tyagi, S., Stevens, M., Mathew, S.: Java Web Services Architecture. published by Elsevier Science, Amsterdam, ch. 2 (July 2003)
16. Musa, J.D.: Operational profiles in software-reliability engineering. IEEE Software, 14–32 (1993)
17. Parnas, D.: On a "Buzzword": Hierarchical structure. In: Proc. of IFIP Congress (1974)
18. Rodrigues, G.N., Rosenblum, D.S., Uchitel, S.: Using Scenarios to Predict the Reliability of Concurrent Component-Based Software Systems. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, Springer, Heidelberg (2005)
19. Trivedi, K.: Probability and Statistics with Reliability, Queuing, and Computer Science Applications. J. Wiley and S., Chichester (2001)
20. Uchitel, S., Kramer, J., Magee, J.: Synthesis of Behavioral Models from Scenarios. IEEE Transactions on Software Engineering 29(2) (2003)

21. Yacoub, S., Cukic, B., Ammar, H.: Scenario-Based Reliability Analysis of Component-Based Software. In: ISSRE 1999. Proc. of the 10th International Symposium on Software Reliability Engineering, pp. 22–31 (1999)
22. Yue, K.: Generating interesting scenarios from system descriptions. In: Proc. the 1st international conference on Industrial and engineering applications of artificial intelligence and expert systems, pp. 212 - 218 (1988)
23. http://www.oasis-open.org/
24. www.w3.org/2002/ws/

Appendix: Parameters Estimation

Transition Probability "$P(T_{cicj})$" $P(T_{cicj})$ represents the probability that the service s_i maps an element of its ci-th equivalence class to an element of the cj-th equivalence class of s_j (see section 3).

The literature reports formulas for the estimation of the probability of transition between basic elements (e.g. objects, components or services). For example, in [10] Hamlet et al. have defined a formula for the probability of transition from a component to another one, on the basis of the input domain partition of a component into a set of functional subdomains (i.e. a subdomain for each functionality of the system). In [21] the authors have defined this probability with respect to each pair of components, on the basis of a CDG.

Probability of failure on demand "$fail_j(cj)$" $fail_j(cj)$ represents the probability for the service s_j to fail in one execution [19] with respect to its cj-th equivalence class (see section 3). We assume that it can be estimated by supposing that the operational profile of the service with respect to its equivalence classes is uniform. The estimate of $fail_j(cj)$ is outside the scope of this paper, however a rough upper bound $1/N_{nf}$ can be obtained by monitoring the service [3] and observing it being executed for a N_{nf} number of times with no failures. Besides, several empirical methods to estimate COTS failure rates [14] could be also used.

Probability that the service does not mask an error "$P(NM_j \cap I_{c_j})$" Since we assume that a service could fail only if it does not receive an erroneous message (see section 3) $P(NM_j \cap I_{c_j})$ can be easily estimated with the formula introduced by Abdelmoez et al. in [1]. Their formula does not embed the probability of failure on demand of a component. In fact, in [1] the error propagation probability from component A to component B, that are tied trough the connector X, is defined by the function Prob $Prob([B](x) \neq [B](x')|x \neq x')$, where $[B]$ denotes the function of component B, and x is an element of the connector X from A to B.