Model-Driven Dependability Analysis of WebServices

Apostolos Zarras[1], Panos Vassiliadis[1], and Valérie Issarny[2]

[1] Computer Science Department, University of Ioannina, Greece,
{zarras, pvassil}@cs.uoi.gr
[2] INRIA, Domaine de Voluceau, B.P. 105, 78 153 Le Chesnay Cédex, France,
Valerie.Issarny@inria.fr

Abstract. This paper focuses on the development of a principled methodology for the dependability analysis of composite Web services. The first step of the methodology involves a UML representation for the architecture specification of composite Web services. The proposed representation is built upon BPEL and introduces necessary extensions to support the second step of the methodology, which comprises the specification of properties, characterizing the failure behavior of the elements that constitute the composite Web services. The automated mapping of this extended UML model to Block Diagrams and Markov models is introduced as the third step of the methodology. A comparative analysis of the aforementioned dependability analysis techniques in terms of precision and complexity is also performed.

1 Introduction

The Web services architecture is an emerging paradigm for the development of wide-area distributed systems. It aims at the transparent integration of Web applications, based on XML-related standards, which enable the specification of the basic services provided by the applications and the communication with those services.

Until now, quite a lot of research efforts have been made in the field of Web service composition and coordination. In particular, there exist several approaches dealing with the automated composition of Web services into composite ones (e.g., [1,2,3]). Moreover, there has been work towards coordination protocols for the development of secure [1] and transactional [2] Web services. The most common approaches involve specific languages like WSFL[3] and BPEL[4] that can be employed in order to regulate the workflow-like execution, or orchestration of composite Web services.

[1] http://www.ibm.com/developerworks/webservices/library/ws-secure/
[2] http://www.ibm.com/developerworks/webservices/library/ws-transpec/
[3] http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
[4] http://www.ibm.com/developerworks/webservices/library/ws-bpel/

This paper is placed into a different context, involving *the development of a principled methodology for the dependability analysis of composite Web services.* By definition [4], dependability is a quite wide concept, characterized by a number of attributes including *reliability, availability, safety, security, etc..* The basic impairments to the aforementioned properties are *faults*, causing *errors* in the state of the service. Errors, in turn, result in *failures*, *i.e.*, deviations of the delivered service from its standard specification.

For some of the dependability attributes (namely for reliability, availability and safety) there exist probability-based theoretic foundations, enabling *dependability analysis.* The goal of the analysis is to estimate - predict the values of these dependability attributes, based on properties (e.g., failure rate, redundancy, etc.) that characterize the constituents of a composite Web service. Reliability analysis, for instance, aims at estimating the probability that the composite service is correctly provided for a particular time period. Traditional dependability analysis includes techniques that rely on the specification of constraints, describing what is needed to guarantee correct service provisioning (e.g., Block-Diagrams (BDs) and Fault-Trees (FTs))[5]). Sophisticated approaches include techniques based on the specification of Markov models that precisely describe the failure behavior of the elements that constitute the service [6]. In general, the use of traditional dependability analysis techniques is a tedious task, which requires significant time and modelling expertise [7].

Within this context, this paper deals with the following problem: "Given the specification of a composite Web service in a suitable language like BPEL, how can we assess its dependability attributes (esp. its reliability)?". The input to the problem is the specification of the composite Web service in BPEL. The output is a set of measures, explaining how the composite Web service is characterized in terms of its dependability attributes. The steps to follow towards this end can be summarized as follows:

1. *Map BPEL specifications to UML models.* BPEL is a language that follows the XML paradigm. Being such, it is fully suitable for automated parsing and processing, but, too hard to read and write for human beings. Due to the complexity of the overall task of dependability analysis, it is thus quite helpful to map the BPEL constructs to some easy-to-read notation, hopefully graphical, which can act as the blueprint of the overall scenario. UML [5] can play this role [8]. Therefore, our first contribution involves a principled method to map BPEL constructs to UML elements and derive the respective UML models for composite Web service scenarios. The proposed mapping extends the semantics of the recently adopted UML v2.0 standard as opposed to the one proposed in [8], which relies on previous, substantially different, versions of UML.
2. *Extend the UML diagram with properties for dependability analysis.* Such properties (e.g., failure rate, redundancy) describe the failure behavior of the constituents of the composite Web service. Therefore, our second contribution involves the proposal of a set of *tagged values* for the UML elements

[5] http://www.omg.org/cgi-bin/doc?ptc/2003-08-02

that we introduce, specifically intended for dependability analysis. Then, the UML model has to be annotated with these values, before proceeding to the evaluation of the dependability attributes of the composite Web service.
3. *Automatically generate input to traditional dependability analysis techniques, based on the annotated model.* Our third contribution consists of the automated generation of Block Diagrams and Markov models, serving to the assessment of the dependability attributes of the composite Web service. We follow a principled approach for both of the aforementioned techniques. The case of Block Diagrams is rather straightforward, due to their inherent simplicity. Markov models, on the other hand, involve more complicated modeling through a quite voluminous set of transitions from erroneous to correct states.

In the rest of this paper we focus on the reliability attribute. Availability and safety can be handled similarly. The remainder of this paper is structured as follows. In Section 2 we present the two first steps of the methodology. The third step is covered in two sections: in Section 3 we deal with Block Diagrams and in Section 4, we cover the case of Markov models. Section 5 presents the overall assessment of these techniques in terms of precision and complexity. Finally, Section 6 discusses related work and Section 7 concludes our results and discusses issues for future research.

2 UML for the Dependability Analysis of Web Services

The proposed UML representation of BPEL comprises the definitions of a set of stereotypes detailed in Section 2.1. These stereotypes are further associated with additional properties detailed in Section 2.2, which characterize their failure behavior.

2.1 UML Stereotypes for Composite Web Services

Table 1 gives the definitions of the proposed stereotypes. A composite service in BPEL is described in terms of a *process* specified using a homonymous stereotype. A process consists of *activities*, specified using the *activity* stereotype. The execution of an activity relies on Web services, provided by one or more *partners*, modelled in terms of the *partnerLink* stereotype.

The process specification further includes the description of *fault* and *event* handlers (specified using the *faultHandler* and the *eventHandler* stereotypes). A fault handler includes an activity, triggered upon the occurrence of a failure. Note here that failures that are properly handled (even by aborting the process) do not cause the failure of the overall business process. Hence, in the dependability analysis, we only consider faults for which there exist no fault handlers. An event handler is an activity that executes upon the reception of a particular message.

Our representation allows specifying different kinds of *basic* and *structured* BPEL activities. The execution of a basic activity relies on a single Web service.

Table 1. Stereotypes for structuring composite Web services.

Stereotype	**UML Base Class**	**Parent**
process	Activity	NA
activity	ExecutableNode	NA
partnerLink	ObjectNode	NA
variable	DataStoreNode	NA
catchAll, catch	ExceptionHandler	NA
onMessage, onAlarm	AcceptEventAction	activity
compensationHandler	ExceptionHandler	NA
basicActivity	ExecutableNode	activity
invoke	CallAction	basicActivity
receive	AcceptCallAction	basicActivity
reply	ReplyAction	basicActivity
throw	RaiseExceptionAction	activity
wait	AcceptEventAction	activity
empty	Action	NA
sequence	ActivityPartition	activity
switch	DecisionNode	activity
while	PartitionNode	activity
pick, flow	ActivityPartition	activity

On the other hand, a structured activity consists of a set of (basic or structured) activities and prescribes the order of their execution. In other words, it defines a number of control and data flow dependencies. These dependencies are specified in UML using ControlFlow elements (*i.e.*, arrows stating that the target activity is triggered when the execution of the source activity is done) and DataFlow elements (*i.e.*, arrows stating that the target activity accepts input from the source activity). By definition, every basic activity is associated with a *joinCondition* element, *i.e.*, a predicate logic formula, describing requirements on one or more flow dependencies, specified for the activity. The different kinds of basic activities supported by our representation are: (1) *invoke* activities, specifying the synchronous, or asynchronous invocation of a Web service; (2) *receive* activities, describing the reception of request messages that initiate a process; (3) *reply* activities, delineating responses to request messages that were previously received during the execution of receive activities.

The different kinds of structured activities supported by our representation are: (1) *sequence* activities, consisting of activities that execute sequentially; (2) *switch* activities, consisting of ordered activities associated with conditions - only the first activity whose condition evaluates to true actually executes; (3) *while* activities, comprising a single activity that executes more than once; (4) *pick* activities, consisting of one or more event handlers; (5) *flow* activities, comprising one or more activities, which by default execute concurrently - although, there may exist control and data flow dependencies between the activities, imposing a certain execution order.

2.2 Failure Behavior Properties

The basic properties, characterizing the stereotypes defined in Section 2.1 are given in Table 2. In particular, the process stereotype is associated with measures, which correspond to the basic dependability attributes. The *reliability measure* we use is the probability that a composite Web service executes correctly for a given time period (specified using the *time* property). Similarly, the *availability measure* we consider is the probability that the composite service executes correctly at a given moment in time. As a *safety measure* we assume the probability that there will be no catastrophic failures for a given time period. The impairments to the aforementioned measures are the faults and the failures of the partners that provide the Web services used in the activities of the composite Web service. Faults appear with a certain rate specified using the failure-rate property given in Table 2. Moreover, the *partnerLink* stereotype is associated with properties that allow distinguishing between different kinds of faults and failures. A list of such properties given in Table 2 (detailed definitions can be found in [4]). With respect to the aforementioned properties we distinguish between:

- *Permanent faults i.e.*, faults that are present for the lifetime of a partner. The presence of these faults does not depend on the internal condition of the partner, neither on the external interaction of the partner with the environment.
- *Temporary faults*, *i.e.*, faults that are present for a limited time period. Temporary faults are further divided in: (1) *Transient faults*, *i.e.*, temporary external faults, resulting from the interaction of the partner with the environment. Transient faults disappear with a certain rate (specified using the disappearance-rate property, given in Table 2). (2) *Intermittent faults*, *i.e.*, temporary internal faults, resulting from the interference between the different parts of the partner. Intermittent faults may be either *active* or *benign*. In the former case, the failed partner provides incorrect services, while in the latter the previous does not hold. Intermittent faults repeatedly go from active to benign and back to active with certain rates (specified using the active-to-benign-rate and the benign-to-active-rate properties defined in Table 2).

Note here that the faults and the failures of the underlying middleware infrastructure used for implementing Web services may also be considered as impairments to the dependability of composite Web services [9]. Dealing with the aforementioned issues is out of the scope of this paper. However, it would imply associating the stereotypes we defined for basic and structured activities with properties that are similar with the ones defined for the *partnerLink* stereotype.

The *partnerLink* stereotype is further associated with properties, characterizing the fault tolerance technique that may be used in a partner. More specifically, a partner may represent a redundancy schema, *i.e.*, a composite partner that encapsulates a configuration of redundant partners, which behave as a single

Table 2. Properties of the UML stereotypes.

Stereotype	**Dependability Properties** (Tagged Values)
process	reliability, availability, safety : 0..1
	time : Integer
partnerLink	**Faults**
	nature : {physical, human}
	boundary : {internal, external}
	phase : {design, operational}
	persistence : {permanent, temporary}
	failure-rate : Real
	disappearance-rate : Real
	active-to-benign-rate : Real
	benign-to-active-rate : Real
	Failures
	domain : {time, value}
	perception : {consistent, inconsistent}
	Redundancy
	error-detection : {vote, comp, acceptance}
	execution : {parallel, sequential}
	confidence : {absolute, relative}
	service-delivery : {continuous, suspended}
	$no_{partners}$, $no_{failures}$: Integer
basicActivity	completion-rate : Real
while	no-iter : Integer
pick, switch	branch-prob : Array[]

fault tolerant unit. This schema is characterized by the error detection mechanism used, the number of partners ($no_{partners}$) that constitute it, the number of partner failures that can be tolerated ($no_{failures}$), etc.

For the dependability analysis of composite Web services we must further account for the *completion-rate* of basic activities. The completion rate of structured activities is a function of the completion rates of the basic activities that constitute them (we assume here that there is no additional overhead introduced by structured activities for the coordination of their constituents). For the particular case of *while* activities a property of type integer, named *no-iter*, is associated with the corresponding stereotype. The values of this property represent the approximate number of iterations performed by the *while* activities. Finally, for the case of *switch* (resp. *pick*) activities with N branches we assume a corresponding array of N probabilities, named *branch-prob*, as a property of the *switch* stereotype.

In order to motivate the discussion, in this section we introduce a reference example of composite Web services, chosen out of the BPEL standard specification [6]. The example is given in Figure 1 and involves a loan approval service. The composite service is orchestrated as follows: The *rcvCustomer* activity receives a loan request from a *customer*. Then, a service provided by an *assessor* partner is invoked within the *invAssessor* activity to assess the loan request. This

[6] More examples, technical details, and a detailed discussion regarding Fault Trees can be found in the long version of this paper, in the following URL: http://www.cs.uoi.gr/∼zarras/papers/doa04-long.pdf

invocation results in a report, which is given as input to the *approver* partner, through the *invApprover* activity. The approver then makes a final decision. The decision is sent back to the customer as a reply to his/her loan request through the *rplCustomer* activity.

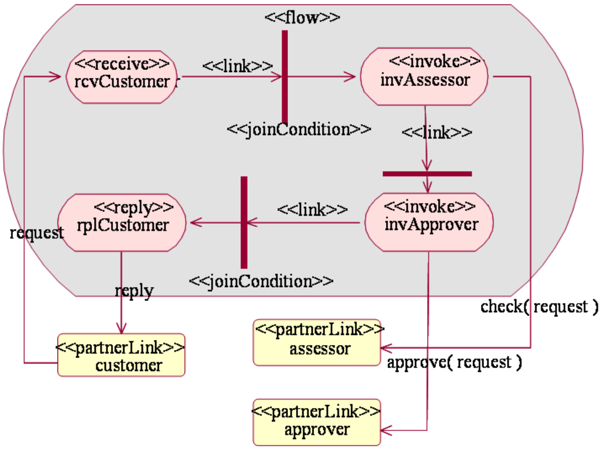


Fig. 1. The architecture of the *loan approval* composite Web service.

3 Automated Transformation of UML Diagrams to Block Diagrams

In this section, we discuss Block Diagrams and the mapping of UML models to these modelling techniques. We use a BD to represent graphically a constraint for correctly providing a composite Web service. Hereafter, we call such a constraint, *constraint-to-succeed.* Roughly, the BD consists of blocks (*i.e.*, boxes), representing the partners that provide the basic Web services, which are executed in the activities of the composite Web service. Those blocks are connected using serial connections. Depending on the different kinds of activities that constitute the composite Web service we build the BD as follows:

(1) For every *sequence*, *flow* or *while* activity A, consisting of the $\alpha_1, \alpha_2, \ldots \alpha_N$ constituent activities, all of them are needed to successfully complete A (for *while* activities $N = 1$).

(2) For every *switch* or *pick* activity A with N branches we have N constituent activities $\alpha_1, \alpha_2, \ldots \alpha_N$. Any of them may execute, depending on the switch condition or the particular events that occur at runtime. Hence, all the constituent activities are needed with a certain probability $branch - prob_{\alpha_i}, i = 1, \ldots N$ to successfully complete A.

(3) Given the above, for every basic activity $\alpha_1, \alpha_2, \ldots \alpha_L$ of A ($L < N$) we have:

(3.a) Let $p_1, p_2, \ldots p_K$ be the non-fault-tolerant partners (*i.e.*, those that do not represent a redundancy schema (see Section 2.2)) that provide basic Web services used in $\alpha_1, \alpha_2, \ldots \alpha_L$. We create a new block for every such partner. The blocks are connected with serial connections. Moreover, the corresponding constraint to succeed is:

$$Success \equiv Success \bigwedge_{l=1,\ldots K} p_l$$

(3.b) Let $p_{K+1}, p_{K+2}, \ldots p_M$ be the fault tolerant partners (*i.e.*, those that represent a redundancy schema) that provide basic Web services used in $\alpha_1, \alpha_2, \ldots \alpha_L$. We create a new i out-of $no_{partners}$ parallel connection for every $p_l, K + 1 < l < M$, where $i = no_{partners} - no_{failures}$ (see Section 2.2 for the semantics of $no_{partners}$ and $no_{failures}$) In the case where $no_{failures} = no_{partners} - 1$, the corresponding constraint to succeed is:

$$Success \equiv Success \bigwedge_{l=K+1,\ldots M} (\bigvee_{i=1,\ldots no_{partners}} p_{l_i})$$

(4) For every composite activity $\alpha_{L+1}, \alpha_{L+2}, \ldots \alpha_N$ of A repeatedly follow steps 1-4.

Based on the BD built for the composite Web service we calculate the values of service's dependability measures through simple combinatorial calculations, involving the dependability measures of the individual partners used in the BD (possibly multiplied by certain branch probabilities if the partners are used within *pick* or *switch* activities).

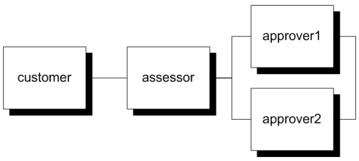


Fig. 2. A block diagram for the loan approval service.

Taking an example, the provision of the loan approval service for a time period t requires using three partners: the *customer*, the *assessor* and the *approver*. Let us assume that the approver is a redundancy schema that consists of two redundant elements and tolerates one failure. The constraint-to-succeed for this case is: $Success_{loan-approval} \equiv customer \wedge assessor \wedge (\vee_{i=1..2} approver_i)$. The corresponding BD is given in Figure 2. The approver is represented by a 1 out-of 2 parallel connection, which is connected in serial with the assessor. For the case of the reliability attribute, the value of the corresponding measure is the probability that $Success_{loan-approval}$ holds for the time period t. This value is obtained using the reliability measures, $R_{customer}, R_{assessor}, R_{approver_1}, R_{approver_2}$, as follows:

$$\begin{aligned} R_{BD} &= P(Success_{loan-approval}) \\ &= R_{customer} * R_{assessor} * (R_{approver_1} + R_{approver_2} - R_{approver_1} * R_{approver_2}) \end{aligned}$$

Every partner p is characterized by a failure-rate, λ_p (see Table 2 in Section 2.2). In computer systems it is typical to assume that the probability that a system fails at a time $T < t$ is exponentially distributed [6]. Given that the partners of a composite service are, in principle, autonomous systems, we can assume that their probability of failure is also exponentially distributed. Then, we have: $R_p = 1 - P_p(t > T) = \exp(-\lambda_p * t)$

4 Automated Transformation of UML Diagrams to Markov Models

Block Diagrams are very simple modeling techniques with the advantage of being easy to construct and quick to compute. Nevertheless, we can do better in terms of precision of our analysis by employing Markov models. In this section, we present Markov models as a general technique for the dependability analysis of composite systems. Then, we focus on our specific problem that involves composite Web services and based on the difficulty of specifying certain parts of a Markov model, we offer automated techniques for these tasks.

4.1 Markov Models

A Markov model for a composite Web service consists of a set of transitions between states of the service. A *state* describes a situation where either the service is correctly provided, or not. In the latter case, we say that the composite Web service is in a *death-state*. The state of the service depends on the situation of the basic activities (which may be encapsulated in structured activities or not) and the situation of the partners that constitute it. The structured activities that encapsulate basic activities do not affect the situation of a composite service as they are not involved in performing any serious computation. They mainly serve as a structuring mechanism, which further determines the execution order of certain encapsulated activities. All the necessary computation for achieving the composite Web service is performed by the services that are invoked within the basic activities that constitute the structured ones. Hence, a state can be seen as a composition of sub-states, representing the partners and the basic activities of a composite service. A basic activity may be in 4 different states: *inactive, active, complete, or failed.* The range of the different state situations for a partner depends on the kind of faults that exist for this partner. A partner with a permanent, or a transient fault may be: *operational, or failed.* Similarly, a partner with an intermittent fault may be: *operational, failed-active, or failed-benign.* The range of the different state situations for partners that represent redundancy schemas further depends on the number of redundant partners and the number of failures that can be tolerated. For example, for a redundancy schema with 2 partners, tolerating 1 failure caused by a permanent, or a transient fault we have three possible situations: *both partners operational, 1 partner failed, both partners failed.*

In general, a Markov model for composite Web services comprises the following different kinds of transitions:

- *Transitions for partner failures*, which take the service from a state where the partner is operational, to a state where the partner is failed. These transitions are characterized by the failure-rate, or the benign-to-active-rate of the partner (see Table 2).
- *Transitions for partner recovery*, taking the service from a state where the partner is failed, to a state where the partner is operational. These transitions are characterized by the disappearance-rate, or the active-to-benign-rate of the partner (see Table 2).
- *Transitions for activity activation*, which take the service from a state where the activity is inactive, to a state where the activity is active. These transitions take place only if the preceding activities are complete (*i.e.*, the join condition of the activity holds) and the partner used by the activity is not failed.
- *Transitions for activity completion*, taking the service from a state where the activity is active, to a state where the activity is complete. These transitions are characterized by the completion-rate of the activity (see Table 2).
- *Transitions for activity failure*, taking the service from a state where the activity is active, to a state where the activity is failed due to the failure of a partner used by the activity. These transitions are characterized by the failure-rate, or the benign-to-active-rate of the partner (see Table 2).

The value of the reliability measure equals to the probability of reaching a death-state of the Markov model within a given time period t. The calculation of this value involves solving a system of first-order differential equations [6].

4.2 The General Framework for the Automated Generation of Markov Models

Generally, it is recognized that the specification of Markov models is a complex and error-prone task [7]. To deal with this problem Johnson [7] proposed an algorithm that relies on the concepts discussed in the previous subsection. In particular, states are modelled as tuples of *integer values*, representing elements that provide basic services. The algorithm generates Markov models, given the following input:

- The definition of the range of tuples that constitute the Markov model. The range definition is given as a tuple of *integer variables*. Each variable represents the range of all possible state situations for an element.
- The definition of a death-state constraint for the Markov model, *i.e.*, a conditional statement, defined on the values of the range variables. This statement evaluates to true for tuples that represent the death-states of the Markov model.
- The definition of an initial state for the Markov model.

The algorithm further accepts as input transition rules between *sets* of Markov states. A transition rule consists of a *conditional statement* and a *transition statement*. The *conditional statement* is defined on the values of the range variables and identifies a set of source states that have common features (e.g., states where a particular partner π is operational). From all these source states there should be transitions to target states, which also share common features (e.g., π is failed in all target states). Moreover, the transitions to the target states are characterized by a common rate (e.g., the failure-rate that characterizes π). The *transition statement* of the rule specifies this common rate and the common features shared amongst the set of the target states.

Given the above, the algorithm starts from the initial Markov state and recursively applies the transition rules, as long as, their conditional statements hold for this state. During a recursive step and for a particular transition rule, the algorithm produces a transition to a state derived from the initial one. If the death-state constraint holds for the resulting target state, the recursion stops. That way the algorithm automatically produces all the possible state transitions for the Markov model.

Still, the specification of the transition rules is complicated and error-prone, especially for the case of complex, composite Web services. To completely alleviate the problem of specifying complex Markov models for the dependability analysis of composite Web services we propose *an automated procedure that generates input models for Johnson's algorithm, from UML models of composite Web services*. The generated models are then given as input to the ASSIST tool [10], which implements Johnson's algorithm and generates a complete Markov model. Finally, the Markov model may be given as input to tools like the SURE reliability analysis tool [6], which solve Markov models and calculate the values of dependability measures.

4.3 Generating State-Range Definitions

The generation of a state range definition from the architectural description of a composite Web service relies on the following steps. First, we select all the partnerLink elements used in the specification of the process that describes the architecture of the composite service. Each one of them represents a partner and a corresponding variable is created in the state-range definition. The range of the integer values for this variable depends on the fault and the failure properties that are associated with the partnerLink element. More specifically we have:

– For a partner π with permanent, or transient faults the value of the variable is 0 in states where π is operational and 1 in states where π is failed.
– For a partner π with intermittent faults the value of the variable is 0 in states where π is operational, 1 in states where π is failed and the fault is active and 2 in states where π is failed and the fault is benign.

Moreover, the range of the integer values depends on the redundancy properties that are associated with the partnerLink element.

Following, we select all the basic activities (invoke, receive, and reply activities) that are specified in the process (as in previous cases, some of them may be encapsulated into more complex structured activities). For each activity α, we also create a variable in the state-range definition. More specifically, if α is encapsulated in a *while* structured activity that performs approximately *no-iter* iterations, the variable takes values from 0 to *no-iter*+1. Otherwise, the variable takes values from 0 to 3. The semantics of these values are summarized in Table 3.

Table 3. Range of state-range variables for activities.

embedded in *While* activities		**not embedded in *While* activities**	
Value	**Semantics**	**Value**	**Semantics**
0	α is inactive.	0	α is inactive.
$i : 1, \ldots,$ *no-iter* - 1	α is executed for the i_{th} time.	1	α is active.
no-iter	α is complete.	2	α is complete.
no-iter+1	α is failed due to the failure of the partner used by this activity.	3	α is failed due to the failure of the partner used by this activity.

Based on the previous steps, the state-range definition for the loan approval service is:

$$\text{space} = (\mathit{customer} : 0..1,\ \mathit{assessor} : 0..1,\ \mathit{approver} : 0..1,\\ \mathit{rcvCustomer} : 0..3,\ \mathit{invAssessor} : 0..3,\ \mathit{invApprover} : 0..3,\ \mathit{rplCustomer} : 0..3);$$

Note that the syntax used is the one required by the SURE-ASSIST analysis tools [6], which we use for solving Markov models.

4.4 Generating Death-State Constraints

A process is considered as failed in states where any of the activities that constitute it is failed (see Section 3). Hence, to generate a death-state constraint, we select all the activities of the process. Based on the selected activities, we build the death-state constraint, which is the disjunction of a number of boolean expressions. Each expression involves a state-range variable that represents one of the activities, say α. If α is encapsulated in a *while* activity that performs *no-iter* iteration, the expression evaluates to true if the variable equals to *no-iter*+1 (see Table 3). In all other cases the expression evaluates to true if the variable equals to 3. Following the aforementioned steps for the loan approval service gives us the following death-state constraint:

$$\text{deathif } (\mathit{rcvCustomer} = 3 \lor \mathit{invAssessor} = 3 \lor\\ \mathit{invApprover} = 3 \lor \mathit{rplCustomer} = 3);$$

4.5 Generating Transition Rules

The generation of transition rules from the UML model of a composite service is slightly more complicated. In Section 4.1 we identified 4 different categories of transitions. Consequently, here we distinguish between 4 different categories of transition rules.

Transition rules for partner failures: for every partner specified in the process that describes the composite Web service, we generate a rule whose conditional statement holds for all source states where the partner is operational and there are no active activities. The rule prescribes that for these source states there should be transitions to target states, whose common feature is that the partner is failed. The rate for these transitions is the failure-rate, or the benign-to-active-rate of the partner. Following, we give an example of a rule for the approver partner of the loan approval service:

if $approver = 0 \wedge invApprover \neq 1$ then
tranto $(customer,\ approver + 1,\ rcvCustomer, invApprover, rplCustomer)$
by failure-rate; endif;

Transition rules for partner recovery: for every partner that may fail because of *temporary* faults (i.e., transient, or intermittent faults) we generate a corresponding transition rule. The conditional statement of this rule holds for states where the partner is failed. On the other hand, the transition statement depends on the kind of the fault that may occur for the partner. For transient faults, in particular, the transition statement specifies transitions to target states, whose common feature is that the partner is operational again. The rate for these transitions is the disappearance-rate that is associated with the partner. For intermittent faults, the transition statement prescribes transitions to target states, whose common feature is that the partner is still failed, but the fault is not active. The rate for these transitions is the active-to-benign-rate that is associated with the partner. Below, we give an example of a rule that is generated for the approver partner of the loan approval service, if we suppose that the approver may fail because of a transient fault.

if $approver = 1$ then tranto $(approver = 0)$ by disappearance-rate; endif;

Transition rules for activity activation: for every basic activity α of the process we generate a transition rule, whose conditional statement holds for states where: (1) The activity is inactive; (2) The activities upon which α depends are complete (*i.e.*, the join condition of α holds); (3) The partner, used in α is operational.

Hence, to build the conditional statement we rely on the dataflow and control dependencies that are specified for α and the join condition that is associated with it. The transition statement of the rule states that for all source states there should be transitions to target states, whose common feature is that α is active. If α is embedded in a *pick* or a *switch* activity, the transitions are characterized by the *branch-prob* of the corresponding branch. Otherwise, they are characterized by a default rate. If α is encapsulated in a *flow* activity together with activities $\beta, \gamma, \ldots$ and the join condition for all of them is the same, then the conditional and the transition statements of the rule involve all these activities, which are finally concurrently activated. Below, we give an example of a rule for the activation of the invApprover activity that uses the approver partner and depends on the completion of the invAssessor activity.

$$\text{if } invApprover = 0 \wedge invAssessor = 2 \wedge approver = 0 \text{ then}$$
$$\text{tranto } (invApprover = 1) \text{ by trig-rate; endif;}$$

Transition rules for activity completion: for every basic activity α we further generate a transition rule, whose conditional statement holds for states where: (1) α is active; (2) The partner that is used in α is operational. Regarding the transition statement of the rule we have: (1) If α is encapsulated in a *while* activity, then the transition statement prescribes transitions to target states whose common feature is that α is reactivated. These transitions actually model the fact that in the source state α executes within the i_{th} iteration of the while activity, while in the target state it executes within the $i_{th} + 1$ iteration of the while activity (2) Otherwise, the transition statement specifies transitions to target states, whose common feature is that α is complete. The transitions described by the aforementioned rule are characterized by the completion-rate of α. Following, we give an example of a rule for the completion of the invApprover activity that uses the approver partner.

$$\text{if } invApprover = 1 \wedge approver = 0 \text{ then}$$
$$\text{tranto } (invApprover + +) \text{ by completion-rate; endif;}$$

Transition rules for activity failure: for every basic activity α we generate a transition rule, whose conditional statement holds for states where: (1) α is active. (2) The partner that is used in α is operational. The transition statement of the rule states that there should be transitions from the aforementioned states to target states, whose common feature is that the partner is failed. In the target states, α is also considered as failed due to the failure of the partner. The rate for these transitions is the failure-rate, or the benign-to-active-rate of the partner. Following, we give an example of a rule for the approver partner of the loan approval service:

$$\text{if } approver = 0 \wedge invApprover = 1 \text{ then}$$
$$\text{tranto } (customer, assessor, approver + 1, rcvCustomer,$$
$$invAssessor, invApprover + 2, rplCustomer) \text{ by failure-rate; endif;}$$

5 Assessment

To assess the overall methodology proposed in this paper, we experiment using the loan approval Web service. In particular, we assess the advantages and the disadvantages of BDs and Markov models regarding their precision and their complexity in the context of Web services.

In terms of our motivating example, we used the following setup for our experiments. The approver partner in the loan approval service fails due to a transient fault. The rest of the partners in the loan approval service fail because of permanent faults. For all kinds of faults we consider a range of failure-rates from 10^{-10} to 10^{-3}. Transient faults have a disappearance rate of 0.5. Following, we generated the BDs and the Markov models for the loan approval service. Based on the Markov models, we used the SURE tool to calculate the values of the reliability measure, for each different failure-rate and for a period t of 10 time-units. Similarly, we calculated corresponding values of the reliability measure using the BDs. Figure 3 (a) shows the results we got. More specifically, we observe that the reliability values that were calculated using the BDs are smaller compared to the ones calculated using the Markov models. The percentage of the difference between the two dependability analysis techniques increases as we increase the failure rates (see Figure 3(b)). The BD-based technique underestimates the reliability of the service because it does not take into account that the transient faults of partners may disappear before the activation of the activities that use those partners. In these cases, the transient faults do not affect the correct execution of the service. This fact is captured in the Markov models, which specify more precisely the failure behavior of the elements that constitute the composite Web services.

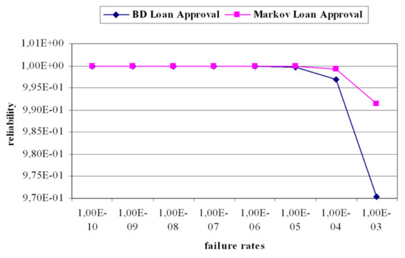
Based on the above, it is safe to argue that, as originally expected, the Markov-based technique is more precise for analyzing the dependability of composite Web services. However, as we discussed in Section 4, the most serious argument against its use is that the specification of Markov models is a complex task. A reasonable experimental metric for the complexity of a Markov model, C_{markov}, is the number of transitions rules required as input to Johnson's algorithm [7]. Moreover, the complexity of a BD, C_{BD} can be measured with respect to the number of blocks that constitute it. Then, for a composite Web service that consists of N partners with permanent faults, M partners with temporary faults and K activities, we have: $C_{Markov} = N+2*M+3*K$ and $C_{BD} = N+M$.

For the specific case of the loan approval we have the values given in Figure 3(c). Moreover, BDs are quite faster to compute due to their simplicity. Figure 3(d) shows the execution times for calculating the values of reliability, given the generated BDs and Markov models.

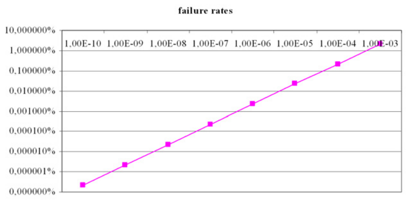
The previous results, generally, highlight the significance of principled methodologies for the automated transformation of Web service architectural specifications, into traditional models (Markov models, queueing networks, Petri-nets, etc.) for the quality analysis of non-functional properties. Another important aspect that advocates the previous argument without being highlighted by the numbers shown in Figure 3 is that these methodologies encapsulate the modelling expertise of domain experts, which is not necessarily part of the knowledge of everyday's developers.

6 Related Work

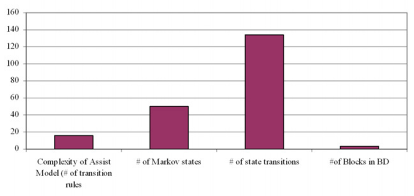
The issue of dependability analysis for conventional composite systems has been explored in the past [11,9,12,13,14]. There are both similarities and differences with this line of research. On the common side, we share the methodological



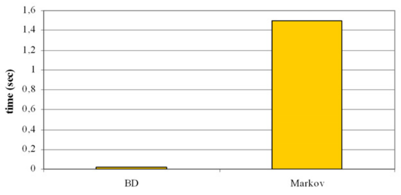
(a) Reliability for the shipping service.



(b) % of reliability underestimation of BDs.



(c) Complexity of Markov models and BDs



(d) Execution times

Fig. 3. Experimental evaluation with BD's and Markov models

approach to the problem (*i.e.*, given a certain input, we map it to UML - for ease of modelling - and then we transform the UML diagram to a well known dependability analysis technique). On the other hand, there are prominent differences, specifically tailored for the case of Web Services: both the input (BPEL in our case) and the automated derivation of the dependability analysis models are different.

In the context of composite Web services, the issues of quality specification, analysis and management just begin to gain the attention of various research communities. More specifically, in [15] the authors propose a framework for the provision of differentiated levels of service that meet the customers' functional and quality requirements, described in terms of Service Level Agreements (SLAs). SLAs are specified using a declarative language, named WSLA. SLAng is also a language for the specification of SLAs [16]. While these approaches are quite generic, we specifically focus on dependability properties and dependability analysis techniques. The dependability properties can be seen as SLA attributes. In [17] the authors also propose an infrastructure-based solution for the provision of differentiated levels of service. They particularly deal with performance SLA attributes. In [18] the authors deal with a similar problem. More specifically, in this approach the input is the specification of a composite Web service that combines N primitive Web services. Moreover the authors assume the existence of N sets of compatible primitive Web services characterized by a number of quality attributes like reliability, performance, price, reputation, etc. Then, they propose an analysis method that allows selecting N services out of the N sets, which can be used in the composite Web service to achieve optimal quality. Although the proposed approach is interesting, the part of the analysis that concerns reliability is simplified and can be enhanced based on the principled methodology we propose in this paper.

7 Conclusions

In this paper, we investigated the dependability analysis of composite services. More specifically, we concentrated on a principled methodology for achieving the previous. First, we presented a representation for the architecture specification of composite Web services that relies on UML v2.0 and introduces necessary extensions to support the specification of properties that characterize the failure behavior of the elements that constitute the composite Web services. Then, we detailed the automated transformation of UML models that rely on the aforementioned representation to Block Diagrams and Markov models, which enable the subsequent estimation of the reliability measures of the composite Web services. Finally, we performed a comparative analysis of the BDs and Markov models with respect to their precision and complexity. This analysis revealed some of the benefits of the proposed methodology.

A more detailed evaluation in the context of real-world case studies is considered as part of our future research. Another interesting topic for future research is that of building a quality analysis Web service that performs the dependability analysis, *on-the-fly*, by monitoring a set of available Web services; such functionality can be either part of a UDDI implementation, or part of a middleware infrastructure that supports the development of composite Web services. In the latter case, the complexity of the analysis techniques plays a even more important role, especially if the infrastructure is targeted to the development of Web services in pervasive computing environments [19].

Acknowledgements. This paper was partially supported by the European Commission and the Greek Ministry of Education through EPEAEK II and by the IST programme of the European Commission FET, under the IST-2001-32645 DBGlobe project.

References

1. D. Florescu, A. Grunchagen and D. Kossmann: XL: An XML Language for Web Service Specification and Composition. In: Proceedings of 11th ACM International Conference on the World Wide Web (WWW'02). (2002)
2. J. Yang and P. Papazoglou: Web Component: A Substrate for Web Service Reuse and Composition. In: Proceedings of CAISE'02. (2002)
3. B. Medjahed, A. Bouguettaya and A. Elmagarmid: Composing Web Service on the Semantic Web. VLDB Journal **12** (2003) 333–351

4. J-C. Laprie: Dependable Computing and Fault Tolerance: Concepts and Terminology. In: Proceedings of the 15th International Symposium on Fault-Tolerant Computing (FTCS-15). (1985)
5. NASA: Reliability Block Diagrams and Reliability Modeling. Technical report, NASA Glenn Research Center (1995) http://www-osma.lerc.nasa.gov/rbd/rbdtut.html.
6. R. W. Butler: The SURE Approach to Reliability Analysis. IEEE Transactions on Reliability **41** (1992) 210–218
7. S. C. Johnson: Reliability Analysis of Large Complex Systems Using ASSIST. In: Proceedings of the 8th AIAA/IEEE Digital Avionics Systems Conference. (1988) 227–234
8. K. Mantell: From UML to BPEL. Technical report, IBM (2003) http://www-106.ibm.com/developerworks/webservices/library/ws-uml2bpel/.
9. V. Issarny, C. Kloukinas and A. Zarras: Systematic Aid for Developing Middleware Architectures. Communications of the ACM (CACM) **45** (2002) 53–58
10. S. C. Johnson and D. P. Boerschlein: ASSIST User Manual. NASA Langley Research Center. (2000)
11. A. Zarras and V. Issarny: Automating the Performance and Reliability Analysis of Enterprise Information Systems. In: Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE'01). (2000) 350–355
12. A. Zarras, C. Kloukinas and V. Issarny: Quality Analysis of Dependable Systems: A Developer Oriented Approach. In: Architecting Dependable Systems. Volume 2677 of LNCS. Springer-Verlag (2003) 197–218
13. G. N. Rodrigues, G. Roberts, W. Emmerich and J. Skene: Reliability Support for the Model Driven Architecture. In: Proceedings of the 2nd IEEE-ACM-SIGSOFT ICSE Workshop on Software Architectures for Dependable Systems (WADS'03). (2003) 7–13
14. I. Majzik, A. Pataricza and A. Bondavalli: Stochastic Dependability Analysis of System Architecture Based on UML Models. In: Architecting Dependable Systems. Volume 2677 of LNCS. Springer-Verlag (2003) 219–244
15. A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer and A. Youssef: Web Services on Demand: WSLA-Driven Automated Management. IBM Systems Journal **43** (2004) 136–158
16. J. Skene, D. Lamanna and W. Emmerich: Precise Service Level Agreements. In: Proceedings of the 26th IEEE/ACM/SIGSOFT International Conference on Software Engineering (ICSE'04). (2004) 179–188
17. V. Cardellini, E. Casalicchio, M. Colajanni and M. Mambelli: Web Switch Support for Differentiated Services. ACM SIGMETRICS Performance Evaluation Review **29** (2001) 14–19
18. L. Zeng, B. Benatallah and M. Dumas: Quality Driven Web Services Composition. In: Proceedings of the 12th ACM International Conference on the World Wide Web (WWW'03). (2003) 411–421
19. V. Issarny, D. Sacchetti, F. Tartanoglou, F. Sailhan, R. Chibout, N. Levy and A. Talamona: Developing Ambient Intelligence Systems: A Solution Based on Web Services. Journal of Automated Software Engineering (2004) (To appear.).