Computing manuscript No.
(will be inserted by the editor)

Dependability Certification of Services: A Model-Based Approach

Claudio A. Ardagna · Ravi Jhawar · Vincenzo Piuri

Abstract The advances and success of the Service-Oriented Architecture (SOA) paradigm have produced a revolution in ICT, particularly, in the way in which software applications are implemented and distributed. Today, applications are increasingly provisioned and consumed as web services over the Internet, and business processes are implemented by dynamically composing loosely-coupled applications provided by different suppliers. In this highly dynamic context, clients (e.g., business owners or users selecting a service) are concerned about the dependability of their services and business processes.

In this paper, we define a certification scheme that allows to verify the dependability properties of services and business processes. Our certification scheme relies on discrete-time Markov chains and awards machine-readable dependability certificates to services, whose validity is continuously verified using run-time monitoring. Our solution can be integrated within existing SOAs, to extend the discovery and selection process with dependability requirements and certificates, and to support a dependability-aware service composition.

Keywords BPEL, Dependability Certification, Markov Chains, Web Services

1 Introduction

The increasing demand for flexibility and extensibility in software reuse and integration has resulted in the wide adoption of web services and SOA applications. The availability of a range of web services published by different providers, coupled with standard XML-based protocols, form a digital ecosystem that allows for the design and dynamic composition of business processes across organization boundaries [26]. While the benefits are immense, this software building paradigm has changed the dimension of risks in business processes, because the failures in partner services are beyond the scope of the business process owner [18, 19, 31]. As a result, clients are increasingly concerned

Dipartimento di Informatica · Università degli Studi di Milano · Crema (CR), 26013, Italy
E-mail: *firstname.lastname*@unimi.it

about service failures that may affect the functional and non-functional properties of their business processes. In this context, trustworthiness of services is a critical factor for clients, and raises the need of adapting current development, validation, and verification techniques to the SOA vision [12, 14]. In particular, the definition of assurance techniques increasing clients confidence that a service complies with their non-functional requirements becomes of utmost importance.

A suitable technique to address the above concerns is based on certification [11]. Originally, certification schemes targeted static and monolithic software, and produced human-readable certificates to be used at deployment and installation time [15, 34]. With the advent of web services, the problem of certifying software systems has been exacerbated and now requires the definition of certification schemes that address the issues introduced by a service-based ecosystem and its dynamics. An interesting line of work is focusing on security certification of services [3, 4], involving definition of approaches that support machine-readable certificates and can be integrated within the service discovery and selection process. However, since other non-functional properties, such as reliability, availability, and safety, are important in this context, we concentrate here on the larger domain of dependability certification. The fundamental requirement for our certification scheme is the ability to verify dependability properties of services and business processes with a given level of assurance, and prove service robustness against failures to the clients.

In this paper, we define a certification scheme that, starting from a model of the service as a Symbolic Transition System (STS) (Section 3), generates a certification model in the form of a discrete-time Markov chain (Section 4). The certification model is used to validate whether the service supports a given dependability property with a given level of assurance. The result of property validation and certification is a machine-readable certificate that represents the reasons why the service supports a dependability property and serves as a proof to the clients that appropriate dependability mechanisms have been used while building it. To complement the dynamic nature of service-based infrastructures, the certificate validity is continuously verified using run-time monitoring, making the certificate usable both at discovery- and run-time (Section 5). Our certification scheme allows clients to select services with a set of certified dependability properties (Section 6), and supports dependability certification of composite services (Section 7).

The contribution of this paper is threefold: *i)* we define a dependability certification scheme for services; *ii)* we provide an approach to service selection that considers clients' dependability requirements; *iii)* we define a solution to the dependability certification of composite services, where the dependability properties of a composite service are calculated on the basis of the dependability certificates of the component services. This paper extends the work in [5] by: *i)* defining an STS-based modeling solution for services under certification, and a process that generates the corresponding certification models; *ii)* providing an enhanced solution to dependability-aware service selection; *iii)* proposing a novel approach to the certification of composite services.

Table 1 Summary of operations realized by eShop partner services

Service	Operation	Description
Vendor	`browseItems`(*query*)	Allows customers to browse available items
	`buyGoods`(*itemID,data*)	Allows customers to buy an item
Shipping	`shipItems`(*itemID,addr*)	Allows customers to ship an item to an address
Storage	`login`(*usr,pwd*)	Provides password-based authentication and returns an authentication token
	`write`(*data,token*)	Stores data in the remote server
	`read`(*query,token*)	Provides access to data stored in the server
	`logout`(*token*)	Allows customers to log out

2 Reference Scenario and Basic Concepts

In this section, we describe our reference scenario and some basic concepts on dependability certification.

2.1 Reference Scenario

Our certification solution considers a highly dynamic and distributed service-based infrastructure that involves the following main parties: *i)* a trusted *certification authority* that certifies the dependability properties of services; *ii)* a *service provider* that implements a service and engages with the certification authority to obtain a certificate for the service; *iii)* a *client* (business owner) that establishes a business relationship with one or more service providers and uses a set of certified services to implement its business process; *iv)* a *service discovery* that enhances a registry of published services to support the certification process and metadata. We note that the client can also be a service consumer searching for and selecting a single certified service.

Our reference scenario is an online shopping service (*eShop*) that allows its customers to browse and compare available items, purchase goods, and make shipping orders over the Internet. The eShop business owner is the client of our framework, who implements eShop as a business process using *i)* three vendor services that offer a range of goods for trade to eShop, *ii)* two shipment services that deliver items to a customer address, and *iii)* an external storage service to store, retrieve, and update shopping information. Table 1 summarizes the details about the operations of partner services.

When a query to browse available items is provided to eShop, a call to operation `browseItems` of all three vendor services is made. The result from each vendor is reported to the customer, say in a tabular form, to enable comparison. The customer can then choose to purchase an item from a specific vendor. In this case, first a call to operation `buyGoods` of that vendor service is made, and then operation `shipItems` of the shipment service with minimum freight cost is invoked. For each transaction, eShop stores the customer, vendor, and shipping specific data using operation `write` of the storage service. Shopping information is then accessed using operation `read` whenever necessary. Operations `write` and `read` are invoked after a successful login.

Intuitively, failures in the partner services may have an impact on eShop and, therefore, in addition to the functional properties, dependability properties of partner services become of paramount importance to eShop. For example, a failure in one of the vendor services may result in quality degradation of eShop, while a failure in the storage service may affect its overall reliability and availability. Hence, eShop must integrate only those external services that satisfy its dependability requirements. In this context, a dependability certificate can serve as an effective means of assurance to the eShop business owner, by providing a proof that its partner services support a given set of dependability properties. A service discovery that provides a selection approach based on dependability certificates can further serve as a means to search and integrate appropriate partner services. For simplicity, whenever not strictly required, we will use a simplified version of the motivating scenario and discuss the concepts in this paper using the storage service.

2.2 Basic Concepts

A service provider implements its service using a set of dependability mechanisms, and engages with the certification authority in a process that certifies the dependability properties of the service. To realize this process, the certification authority must define: *i)* a hierarchy of dependability properties to be certified; *ii)* a model of the service to drive the certification process; and *iii)* a policy to assess and prove that a given property holds for the service.

Hierarchy of dependability properties. According to [6,33], dependability concept usually consists of three parts: the *threats* affecting dependability, the *attributes* of dependability, and the *means* by which dependability is achieved. The threats identify the errors, faults, and failures that may affect a system. The attributes integrate different aspects of dependability and include the basic concepts of availability, reliability, safety, confidentiality, integrity, and maintainability. In this paper, we consider a subset of dependability attributes, that is, availability, reliability, and safety, which measure the ability of the service to be up and running, and to be resistant to failures. Finally, the means define the categories of mechanisms, such as fault prevention, fault tolerance, and fault removal, that can be used to achieve system dependability.

Starting from the above definition of dependability, we define a hierarchy of dependability properties that are the target of our certification scheme. First, we consider the dependability attributes (abstract properties below) to represent the general purpose dependability characteristics of the service under certification. Then, a concrete property p=$(\hat{p}, A)$ enriches the abstract property $p.\hat{p}$ with a set $p.A$ of attributes that refer to the threats the service proves to handle, the mechanisms used to realize the service, or to a specific configuration of the mechanisms that characterizes the service to be certified. We note that the mechanisms represent specific implementations of dependability means. For each attribute $attr{\in}A$, according to its type, a partial or total order relationship $\preceq_{attr}$ can be defined on its domain Ω_{attr}, and $v(attr)$

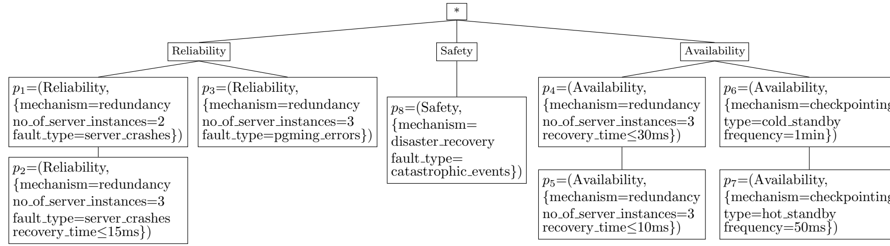


Fig. 1 An example of a hierarchy of dependability properties

represents the value of *attr*. If an attribute does not contribute to a property configuration, its value is not specified. In general, attributes represent a service provider's claims on the dependability of its service. For instance, when *attr* is fault_type, $v(attr)$ can be *crash failure*, *programming error*, or *byzantine failure*.

The hierarchical ordering of dependability properties can be defined by a pair $(\mathcal{P}, \preceq_P)$, where $\mathcal{P}$ is the set of all concrete properties, and $\preceq_P$ is a partial order relationship over $\mathcal{P}$. We note that an abstract property corresponds to a concrete property with no attributes specified. Given two properties p_i,$p_j \in \mathcal{P}$, we write $p_i \preceq_P p_j$ if *i)* $p_i.\hat{p}=p_j.\hat{p}$ and *ii)* $\forall attr \in A$ either $v_i(attr)$ is not specified for p_i or $v_i(attr) \preceq_{attr} v_j(attr)$. The relation $p_i \preceq_P p_j$ means that p_i is weaker than p_j and a service certified for p_j also holds p_i. Figure 1 shows an example of a hierarchy of dependability properties. Each node represents a concrete property p=($\hat{p}$,A). Each child node of a given node represents a stronger property and takes precedence in the hierarchy. For instance, $p_1 \preceq_P p_2$, $p_4 \preceq_P p_5$, and $p_6 \preceq_P p_7$. $p_1 \preceq_P p_2$ since p_2 specifies additional guarantees on the recovery time). We note that some properties are incomparable despite the same abstract property (e.g., p_4 and p_6).

Symbolic Transition System (STS). In our context, the service model must succinctly represent the functional and dependability properties of the service. To this aim, it must represent the different states of the service, the dependability mechanisms, and their specific configurations. Following the approaches in [14, 21, 32], we model services, interactions within a service, and communications between different services using STSs. An STS extends a finite state automaton with variables, actions, and guards to capture the complex interactions in a system. It can be defined as follows.

Definition 1 (Symbolic Transition System) A symbolic transition system is a 6-tuple $\langle \mathcal{S}, s_1, \mathcal{V}, \mathcal{I}, \mathcal{A}, \overset{a,g,u}{\rightarrow} \rangle$, where $\mathcal{S}$ is the set of states in the service, $s_1 \in \mathcal{S}$ is the initial state, $\mathcal{V}$ is the set of (location) internal variables specifying data dependent flow, $\mathcal{I}$ is the set of interaction variables representing operation inputs and outputs, $\mathcal{A}$ is the set of actions, and $\overset{a,g,u}{\rightarrow}$ is the transition relation. Each transition $(s_i, s_j) \in \overset{a,g,u}{\rightarrow}$ between two states $s_i, s_j \in \mathcal{S}$ is associated with an action $a \in \mathcal{A}$ that encapsulates a guard g, defining the conditions on transition, and an update mapping u, providing new assignments to the variables in $\mathcal{V}$.

In the following, when possible, we will refer to the transition relation simply with $\rightarrow$. Differently from [14,21,32], our modeling approach is also used when the real implementation of the service (and its dependability mechanisms) is available. This approach permits to generate fine-grained test cases that can be used to generate the certification model of our solution, and validate the dependability properties against various threats (see Sections 4, 5, and 7).

Policy. A certification scheme must verify and prove that a dependability property p is supported by the service. Proving p is equivalent to validating the implementation of dependability mechanisms used by the service to counteract a given (set of) threat. Based on this observation, we define a policy $Pol(p) \rightarrow \{c_1, \ldots, c_m\}$ that contains all conditions $c_1, \ldots, c_m$ on dependability mechanisms necessary to prove that p holds for the service. We note that while the threats specified in the property drive the certification and testing processes (e.g., by defining a given fault injection model), they are not considered in policy specification. This is due to the fact that policies only include conditions that can be quantitatively measured to validate a given mechanism. Hence, we can define a policy corresponding to each property configuration, where each $c_i \in Pol(p)$ defines a relationship derived by the attributes in $p.A$ and mechanisms used to implement the service. For instance, for property p_2 in Figure 1, a policy can be defined with conditions: c_1:no_of_server_instances$\geq$3 and c_2:recovery_time$\leq$15ms. In this context, a dependability certificate is granted to the service when it satisfies the policy proving that it holds a property p with a given level of assurance against a given set of threats (see Section 5).

3 Service Modeling

The complete modeling of the service under certification represents a fundamental step to realize dependability certification, and serves as the basis for the certification authority to carry out its validation process. In this section, we present a modeling approach that allows the certification authority to generate an STS-based model of a service, based on *i)* the dependability property to be certified and *ii)* the information released by the service provider in the Web Service Description Language (WSDL) interface and/or the Web Service Conversation Language (WSCL) document.

3.1 WSDL-based Model

The WSDL interface is the least set of information that a service provider has to release to publish its service, and specifies the set of service operations and the methods of accessing them. In our context, the WSDL interface is used to define the WSDL-based model of the service, as follows.

Definition 2 (WSDL-based model) Let $\mathcal{M}$ be the set of STS-based service models, a WSDL-based model $M_{wsdl} \in \mathcal{M}$ of a service *ws* is an STS that

consists of a set $\{m_{wsdl}\}$ of connected components, each one modeling a single service operation. Each m_{wsdl} is in turn modeled as an STS $\langle \mathcal{S}, s_1, \mathcal{V}, \mathcal{I}, \mathcal{A}, \rightarrow \rangle$ (see Definition 1), where the number of actions modeling operation input and output is equal to two ($|\mathcal{A}|$=2) and the number of states is at least equal to three ($|\mathcal{S}|\geq$3).

Intuitively, each $m_{wsdl}\in M_{wsdl}$ always includes three states modeling the operation interface as follows: *i)* $s_1\in\mathcal{S}$ is the initial state when no input has been received by the service operation, *ii)* $s_2\in\mathcal{S}$ is the intermediate state when the input is received while the output is not yet generated (i.e., when the operation is being performed), and *iii)* $s_3\in\mathcal{S}$ is the final state when the output has been generated and correctly returned to the client. The set $\mathcal{S}$ of states in m_{wsdl} can be extended to represent the *stateful* implementation of the service operation when its source code is available. In this case, the intermediate state s_2 consists of a number of sub-states as described in Example 1 at the end of this section. Guards g at state transitions model the functional correctness of the service and the specific configuration of dependability mechanisms.

3.2 WSCL-based Model

The WSCL document defines the service conversation as the communication protocol between the clients and the service, and the interactions/ordering between various operations within the service. Given a service conversation, we aim to define the WSCL-based model of the service in the form of an STS. To this aim, we consider connected components $m_{wsdl}\in M_{wsdl}$ as our building blocks, and define a set of modeling operators *op*:$\mathcal{M}\times\mathcal{M}\rightarrow\mathcal{M}$ that take as input two service models and return as output their combination. According to the operators typically defining the WSCL conversation, we first define the set $\mathcal{O}$=$\{\odot,\otimes\}$ of modeling operators *op*, where $\odot$ is the sequence operator and $\otimes$ is the alternative operator. We then recursively apply the modeling operators on the WSCL-based model, using the connected components m_{wsdl} as basic elements, to derive M_{wscl} as follows (see Example 1).

$$M_{wscl} = m_{wsdl} \mid M_{wscl} \odot M_{wscl} \mid M_{wscl} \otimes M_{wscl}$$

The WSCL-based model can be defined as follows.

Definition 3 (WSCL-based model) Let $\mathcal{M}$ be the set of service models, a WSCL-based model $M_{wscl}\in\mathcal{M}$ of a service *ws* is an STS $\langle \mathcal{S}, s_1, \mathcal{V}, \mathcal{I}, \mathcal{A}, \rightarrow \rangle$ (see Definition 1), where $\mathcal{S}$ is the union of all the states of component WSDL-based models integrated using the operators in $\mathcal{O}$=$\{\odot,\otimes\}$, and $\mathcal{A}$ represents the set of service operations involved in the conversation.

Given two WSCL-based models M_1 and M_2 combined using $\otimes$, the initial states of the two models can be unified and represented using a single state (e.g., State s_4 in Figure 2(b) is the common state between operations `read` and `write` combined by operator alternative). Similarly, when the two WSCL-based models are combined using $\odot$, the final state of the first model M_1 and

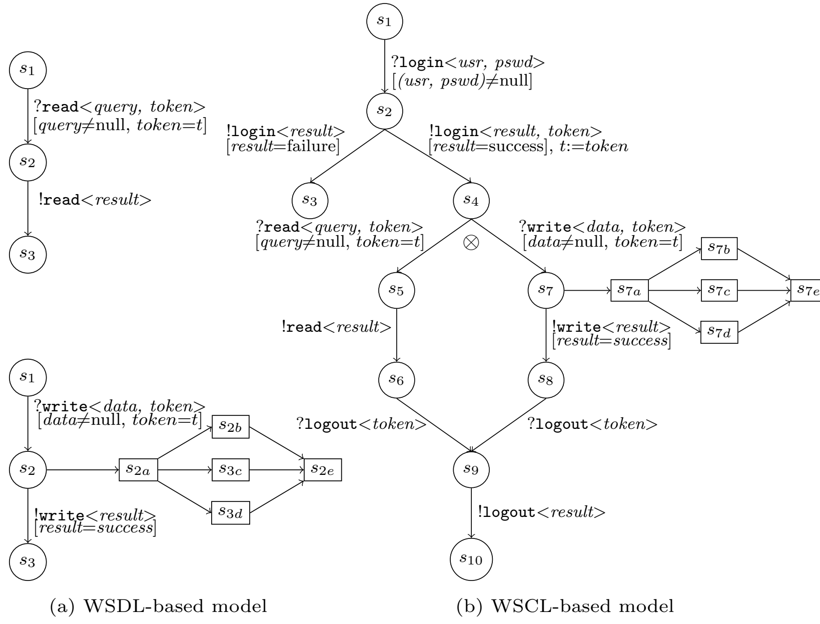


Fig. 2 An example showing the STS-based service models of the storage service. The input actions are denoted as ?`operation`<*parameters*>, while the corresponding output actions are denoted as !`operation`<*results*>. The interface states are represented as circles and stateful implementation states as rectangles

the initial state of the second model M_2 can be represented using a single common state (e.g., State s_4 in Figure 2(b) is the combination of the final state of operation `login` and the initial state of the choice between the operations `read` and `write`). We note that the WSCL-based model resulting from the application of these modeling operators can be further refined to derive a more clear while equivalent representation. For example, the final state of operation `login` in Figure 2(b) can be represented with two branches, where state s_3 is reached as a result of an internal trigger on login failure, and state s_4 represents the final correct state of operation `login`. Similarly to the WSDL-based model, the set $\mathcal{S}$ of states in M_{wscl} can be extended when the source code of the service operations is available. The implementation states of M_{wsdl} are included in M_{wscl} when corresponding interface states are involved in the conversation.

Example 1 Figure 2(a) shows two connected components of the WSDL-based model of the storage service modeling operation `read` with no implementation states (i.e., $|\mathcal{S}|$=3) and operation `write` with implementation states (i.e., $|\mathcal{S}|$>3). We note that, while not shown in Figure 2(a), the model also includes connected components corresponding to operations `login` and `logout`. Let us now consider property p_2 in Figure 1 as the property to be certified for operation `write`. Operation `write` starts at state s_1 waiting for an input. When the input is received and verified to be valid using the guard $data{\neq}$null

and *token*=t, a transition to state s_2 happens (functional correctness is verified). State s_2 contains five sub-states representing stateful implementation of the dependability mechanism, where data, metadata, and index are stored across three redundant storage servers. In particular, state s_{2a} denotes the state when input is provided to servers, states s_{2b}, s_{2c} and s_{2d} represent the state in which data are stored in three different servers, and state s_{2e} performs the output check. A transition from s_{2a} to $s_i \in \{s_{2b}, s_{2c}, s_{2d}\}$ is observed when the i-th server is up and running (i.e., guard [status(server$_i$)=*ok*] is verified), and a transition from s_i to s_{2e} when the i-th server returns *success* (i.e., guard [result(server$_i$)=*success*] is verified). For the sake of clarity, guards validating dependability mechanisms in transitions between states s_{2a} and s_{2e} are not shown in Figure 2. Transition from state s_2 to the final state s_3 happens when *success* is returned by all storage servers.

Figure 2(b) illustrates the WSCL-based model of the conversation that allows the client to access service operation `read` or `write`, after it has been authenticated using operation `login`, and then disconnect using operation `logout`. M_{wscl} is generated by applying modeling operators in $\mathcal{O}=\{\odot, \otimes\}$ on the connected components in M_{wsdl}. The components representing operations `read` and `write` (see Figure 2(a)) are combined using $\otimes$, and then connected to operation `login` using $\odot$. Finally, operation `logout` is appended to operations `read` and `write`. The service starts in state s_1 where it receives the login credentials; if the authentication is successful, it transits to state s_4 and the update mapping assigns the login *token* to the internal variable $t \in \mathcal{V}$. In state s_4, the client can call either operation `read` or `write` with relevant parameters, and perform its task. The client can request to logout from the service in state s_6 or s_8 to reach the final state s_{10}. Set $\mathcal{I}=\{$*usr, pswd, result, token, query, data*$\}$ comprises the state interaction variables.

A service model represents the functional and dependability properties of a service in the form of an STS, and serves as a building block to dependability certification for two reasons. First, it is at the basis of the certification model used to validate a dependability property with a given assurance level. Second, it is used, together with the threats specified in property p to be certified, to generate a set of test cases [35] (service requests) that are used to evaluate dependability behavior of the service and to award a certificate. We note that the more detailed the service model, the more complete and effective the generated test cases, and in turn the higher the certification quality.

4 Certification Model

We define the certification model as a Markov model representation of the service, enabling the certification authority to quantitatively measure the compliance of the service to a property p, and accordingly to award a dependability certificate to the service. Section 4.1 presents the process that receives as input a service model and produces as output a certification model. Section 4.2 defines the concept of assurance level and an approach to calculate it.

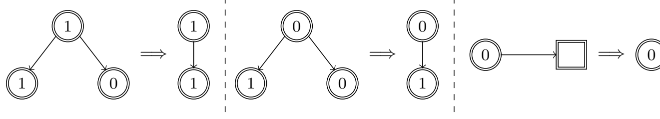


Fig. 3 Pruning rules for interface and implementation states of M^λ. Each three-state component of the service model is denoted with a circle; implementation states with a rectangle

4.1 Markov-based Representation of the Service

The Markov model representation of the service is generated by performing three activities: *i)* prune, *ii)* join states, map policy, and integrate absorbing states, and *iii)* add probabilities.

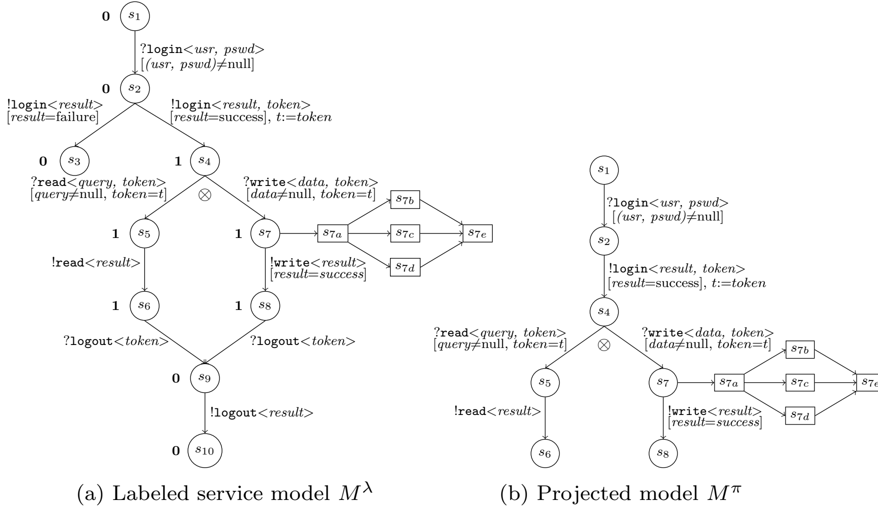
Prune. The prune activity applies a projection π on the service model, over dependability property p, to generate a projected model M^π that *i)* contains all Most Important Operations (MIOs) with respect to p, that is, operations that are needed to certify p, and *ii)* ensures that the projection is consistent with the specifications in the WSDL interface and WSCL document. To obtain the projected model, we introduce a labeling function λ:$\mathcal{S}^I \rightarrow \{0,1\}$, where $\mathcal{S}^I \subseteq \mathcal{S}$ is the set of states in the interface part of the service model, that marks each state $s \in \mathcal{S}^I$ with a binary value $\{0,1\}$. The application of such labeling function results in a *labeled service model* M^λ, which extends the service model M by annotating each state $s \in \mathcal{S}^I$ corresponding to a MIO with 1 and all other states with 0. A state s shared by two or more operations (e.g., State s_4 in Figure 4) is labeled 1 if at least one of these operations is a MIO.

Using labeled service model M^λ, we define a set of *pruning rules* that are used to generate projected model M^π as follows (see Figure 3).

- *Pruning rule for interface states*: It operates recursively on the leaf states in the interface part of the labeled service model and removes those states for which $\lambda(s)$=0. To maintain consistency, if a state s_i has a descendant state s_j for which $\lambda(s_j)$=1, state s_i is not removed even if $\lambda(s_i)$=0.
- *Pruning rule for implementation states*: It removes all implementation states associated with an interface state s for which $\lambda(s)$=0.

We note that, when a WSDL-based model is considered, the pruning rules are applied on each connected component $m_{wsdl} \in M_{wsdl}$ independently, and the component is either removed or taken as it is. The *pruning* activity can then be viewed as a function π that takes a labeled service model M^λ as input, applies the pruning rules, and generates the projected model $M^\pi \in \mathcal{M}$ of the service as output. We note that $M^\pi = \langle \mathcal{S}^\pi, s_1^\pi, \mathcal{V}^\pi, \mathcal{I}^\pi, \mathcal{A}^\pi, \rightarrow^\pi \rangle$ is a sub-model of $M = \langle \mathcal{S}, s_1, \mathcal{V}, \mathcal{I}, \mathcal{A}, \rightarrow \rangle$, such that $\mathcal{S} \subseteq \mathcal{S}^\pi$, $\mathcal{V} \subseteq \mathcal{V}^\pi$, $\mathcal{I} \subseteq \mathcal{I}^\pi$, $\mathcal{A} \subseteq \mathcal{A}^\pi$, $\rightarrow \subseteq \rightarrow^\pi$.

Example 2 Let p=(reliability, {mechanism=redundancy, no_of_server_instances=3,fault_type=server_crashes,recovery_time≤15ms}) be the property to be certified for the storage service and M in Figure 2(b) the WSCL-based model of the service. We first apply the labeling function λ on M to obtain the labeled service model M^λ illustrated in Figure 4(a). We note



(a) Labeled service model M^λ (b) Projected model M^π

Fig. 4 An example of pruning activity applied on the model in Figure 2(b)

that the states corresponding to operations `login` and `logout` are marked 0, since they are not MIOs for the certification of p. Then, following our pruning rules, we obtain the projected service model M^π in Figure 4(b), where operation `logout` has been removed, while a part of operation `login` has been maintained since it has at least a descendant state s such that $\lambda(s)$=1.

Join states, map policy, and integrate absorbing states. This activity applies a transformation $\bowtie$ on the projected service model M^π, over dependability property p, to generate a new model $M^\bowtie$. It is composed of two steps, *i) join states and map policy*, and *ii) integrate absorbing states*, as follows.

The *join states and map policy* step is a manual process that depends on property p, policy $Pol(p)$, service model M, and implemented dependability mechanisms. It first joins the implementation states representing the dependability mechanisms of each service operation in M^π, to model the successful execution flow of service operations. It then uses guards and update mapping on transitions in M^π involving the joined states and, according to policy $Pol(p)$, produces the conditions that regulate transitions in $M^\bowtie$. It finally modifies the interface part of M^π by specifying, for each state transition, a set of conditions derived from g and u. In the following, we denote state transitions as $\stackrel{c_{ij}}{\rightarrow}$, with c_{ij} the conditions on transitions.

The *integrate absorbing states* step inserts two absorbing states C and F, representing the state of correct output and failure, respectively, and connects them to each leaf in the interface part of the model. From the certification point of view, state C is reached when the service satisfies the policy conditions in $Pol(p)$, while state F is reached in case of a policy violation.

The two steps of this activity, together, can be viewed as a function $\bowtie$ that *i)* takes the projected service model M^π as input, *ii)* applies join states

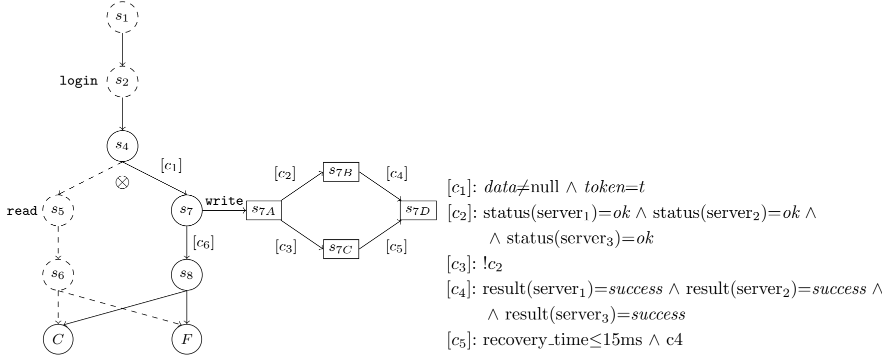


Fig. 5 An example of model $M^{\bowtie}$ of the storage service, obtained after applying join states, map policy, and integrate absorbing states to operation `write` of model M^{π} in Figure 4(b)

and map policy, and integrate absorbing states steps, and *iii)* generates a new model of the service $M^{\bowtie}{=}\langle\mathcal{S}^{\bowtie}, s_1^{\bowtie}, C, F, \overset{c_{ij}}{\rightarrow}^{\bowtie}\rangle$.

Example 3 Let M^{π} in Figure 4(b) be the projected model and p=(reliability, {mechanism=redundancy,no_of_server_instances=3,fault_type=server_crashes, recovery_time≤15ms}) the dependability property. For simplicity, in this example, we consider operation `write` (states s_4, s_7, s_8) only. The portion of model $M^{\bowtie}$ in Figure 5 referring to operation `write` (black lines) is generated as follows. We apply the join states and map policy step on M^{π} over p, to embed $Pol(p)$ within the model. To this aim, we first modify M^{π} using the implementation states of the dependability mechanism redundancy with three server instances (states $s_{7a} - s_{7e}$) and generate the new model states as follows: *i)* state s_{7a} of M^{π} corresponds to state s_{7A} in $M^{\bowtie}$, *ii)* states s_{7b},s_{7c},s_{7d} are represented with states s_{7B} and s_{7C}, and *iii)* state s_{7e} is mapped to state s_{7D}. We then integrate the policy conditions in $Pol(p)$ with guards and update mappings in M^{π} to produce conditions $[c_2]$–$[c_6]$. Here, when the service reaches state s_7, first an implicit transition to the sub-state s_{7A} happens, where it sends the request to three replicated storage servers. The service then moves to state s_{7B} when all servers are fail-free (following condition $[c_2]$); otherwise, when one or more server crashes are detected, it transits to state s_{7C} ($[c_3]$). In s_{7C}, the service starts the recovery process and moves to state s_{7D} if recovery is performed in less than 15ms and all servers return *success* ($[c_5]$). A transition from s_{7B} to s_{7D} is observed if *success* is returned by all the storage servers ($[c_4]$). The service moves to the output state s_8 following condition $[c_6]$ in $M^{\bowtie}$, which is an aggregate of conditions applied at each sub-state of state s_7 ($[c_2]$ to $[c_5]$). We finally integrate the absorbing states C and F and connect them to the final states in the interface part of the model. We note that there is an implicit transition between each state s_i and F (denoted as (s_i, F)) if some unexpected errors or policy violation happen. We also note that if C is reached, a fail-free operation/conversation have been executed.

Add probabilities. The last activity extends $M^{\bowtie}=\langle \mathcal{S}^{\bowtie}, s_1^{\bowtie}, C, F, \stackrel{c_{ij}}{\rightarrow}^{\bowtie}\rangle$ with probability values to generate the Markov chain used in this paper as our certification model M_{cert}. M_{cert} specifies probabilities Pr_{ij} to satisfy the conditions corresponding to each state transition $\stackrel{c_{ij}}{\rightarrow}$, and probability R_i to remain fail-free corresponding to each state s_i. In this context, similar to [10], R_iPr_{ij} represents the probability that execution of a service in state s_i will produce the correct results, and transfer the control to state s_j. The transition from the final state s_k to the correct state C, having probability R_k, is observed if the service satisfies relevant conditions in $Pol(p)$ (with no failures). We note that there is an implicit transition of probability $1-\sum_{j=1}^{k} R_iPr_{ij}$ from each state $s_i \neq s_k$ to F representing a failure or violation of the condition in that state. The transition from s_k to F has probability $1-R_k$. We also note that there can be multiple final states s_k in a WSCL conversation (e.g., s_6 and s_8 in Figure 5). In this case, our Markov model converges all final states to a single node that is then connected to C and F. The transition from one state to another is assumed to follow the Markov property, regardless of the point at which the transition occurs. Our certification model M_{cert} then consists of a state-based, discrete-time Markov chain that combines the failure behavior and system architecture of the service to validate and certify a given set of dependability properties. M_{cert} can be defined as follows.

Definition 4 (Certification model) The certification model M_{cert} of a service is a 6-tuple $\langle \mathcal{S}, s_1, C, F, \stackrel{c_{ij}}{\rightarrow}, R_iPr_{ij}\rangle$, where: $\mathcal{S}=\mathcal{S}^{\bowtie}$ is the set of all states in the model, $s_1=s_1^{\bowtie}$ is the initial state; C is the final correct state; F is the final failure state; $\stackrel{c_{ij}}{\rightarrow}=\stackrel{c_{ij}}{\rightarrow}^{\bowtie}$ represents a transition relation between pairs of states (s_i, s_j) labeled with a condition c_{ij} derived from $Pol(p)$, g, and u; and R_iPr_{ij} is the probability that the service execution provides the correct results, satisfies the conditions in state s_i, and moves to state s_j.

In case a WSDL-based model is used, M_{cert} is produced for each MIO in the service, while a single M_{cert} is generated for a WSCL-based model.

4.2 Assurance Level

We present our interpretation of the Markov model of the service as the assurance level used to quantitatively validate a dependability property p. The certification model can be represented by a transition matrix Q' as follows.

$$Q' = \begin{array}{c} \\ C \\ F \\ s_1 \\ s_2 \\ \vdots \\ s_k \end{array} \begin{array}{c} \begin{array}{cccccc} C & F & s_1 & s_2 & \dots & s_k \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1-\sum_{j=1}^{k} R_1Pr_{1j} & R_1Pr_{11} & R_1Pr_{12} & \dots & R_1Pr_{1k} \\ 0 & 1-\sum_{j=1}^{k} R_2Pr_{2j} & R_2Pr_{21} & R_2Pr_{22} & \dots & R_2Pr_{2k} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ R_k & 1-R_k & 0 & 0 & \dots & 0 \end{pmatrix} \end{array}$$

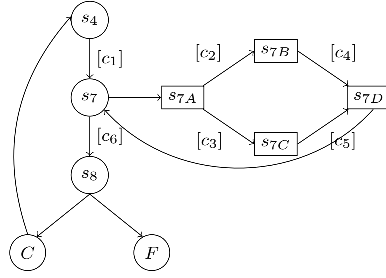


Fig. 6 An example of a certification model for operation `write` in Figure 2(a)

We note that, for the sake of clarity, conditions on transitions are not reported in Q'. The certification authority can use the matrix Q' to estimate the extent to which the service satisfies dependability property p by following the approach presented in [10]. Let Q be a matrix obtained from Q' by deleting rows and columns corresponding to C and F; μ is a matrix such that

$$\mu = I + Q + Q^2 + Q^3 \cdots = \sum_{x=0}^{\infty} Q^x = (I - Q)^{-1}$$

where I is the identity matrix with same dimension as Q. Here, the assurance level of the service is defined as follows.

Definition 5 (Assurance level) Assurance level L of a service deployed in a specific environment is the probability that it satisfies a policy $Pol(p)$ and holds a dependability property $p \in \mathcal{P}$ for a given rate of service executions.

We note that the assurance level characterizes the extent to which a service communication that starts from the initial state s_1 will reach the final execution state s_k, and transit from s_k to the final correct state C. Assurance level L of a service can be estimated using $L = \mu_{1,k} * R_k$, where $\mu_{1,k}$ represents the probability value at 1st row and k^{th} column of the matrix μ, and R_k is the probability of the final execution state to be fail-free. $\mu_{1,k}$ can also be computed using

$$\mu_{1,k} = (-1)^{k+1} \frac{|Q|}{|I - Q|}$$

where $|Q|$ and $|I - Q|$ represent the determinant of Q and $I - Q$, respectively.

When the certification model is generated using the WSDL-based model, assurance level L is calculated for each operation individually. Instead, when the certification model is generated using the WSCL-based model, assurance level of the overall conversation is calculated as a single value.

Example 4 Figure 6 illustrates the certification model corresponding to operation `write` of the storage service in Figure 2(a). This model is obtained after applying prune, join states, map policy, integrate absorbing states and add probabilities to the service model. We note that conditions $[c_1]$−$[c_6]$ are the ones discussed in Example 3. Let the fail-free probabilities of states s_4, s_7, s_8

computed by the certification authority be R_4=0.99, R_7=0.94, and R_8=0.97, and transition probabilities between nodes be Pr_{47}=0.93 and Pr_{78}=0.95. An approach to derive the probability values is discussed in Section 5. For simplicity, in this example, we do not specify the probabilities of internal states $s_{7A}-s_{7D}$, while we use them to deduce probability $R_7 Pr_{78}$. The corresponding transition matrix Q' and matrix $\mu=(I-Q)^{-1}$ are:

$$Q'= \begin{array}{c} C \\ F \\ s_4 \\ s_7 \\ s_8 \end{array} \overset{\begin{array}{ccccc} C & F & s_4 & s_7 & s_8 \end{array}}{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0.0793 & 0 & 0.9207 & 0 \\ 0 & 0.1070 & 0 & 0 & 0.893 \\ 0.9700 & 0.0300 & 0 & 0 & 0 \end{pmatrix}} \qquad \mu= \begin{array}{c} s_4 \\ s_7 \\ s_8 \end{array} \overset{\begin{array}{ccc} s_4 & s_7 & s_8 \end{array}}{\begin{pmatrix} 1 & 0.9207 & 0.8222 \\ 0 & 1 & 0.8930 \\ 0 & 0 & 1 \end{pmatrix}}$$

Here, $\mu_{4,8}$=0.8222. The probability that operation `write` of the storage service satisfies a property p is L=0.8222∗0.97=0.7975.

5 Dependability Certification Process

We design our certification scheme as a two-phase process due to the highly dynamic nature of the SOA environment [17, 20]. The first phase validates the dependability properties of the service before it is actually deployed in a system, and issues a certificate to the service provider based on the initial validation results (*offline phase* in Section 5.1). The second phase monitors the certified properties of the service at run-time, and updates the certificate based on real validation results (*online phase* in Section 5.2). Our certification process results in a dependability certificate life-cycle, which represents the possible states of certificates (Section 5.3). For simplicity, in the following, we assume a certification process that proves and awards certificates with a single dependability property.

5.1 Offline Phase

The offline phase starts when a service provider requests the certification authority to issue a certificate to its service for dependability property p. The certification authority first generates the service model based on p and the specifications released by the service provider. After verifying that the service model conforms to the real service implementation, the certification authority derives the certification model M_{cert} as discussed in Section 4. In our solution, the service model is used to generate service executions (test cases) that are used with M_{cert} to perform property validation. We define a validation function f to verify that the service satisfies policy $Pol(p)$ using M_{cert}, assuming that each condition c_i in M_{cert} is specified as a boolean valued predicate. The service proves to hold p when the conditions in $Pol(p)$ are satisfied with a given level of assurance. The validation function is defined as follows.

Definition 6 (Validation function) The validation function f:$(ws, p, M_{cert}, k) \rightarrow \{true, false\}$ takes service ws under evaluation, dependability property p to be validated, certification model M_{cert} integrating $Pol(p)$, and an index k referring to the service execution that triggers policy verification as input, and returns $true$ when relevant conditions in $Pol(p)$ are satisfied with respect to M_{cert}, $false$ otherwise, as output.

In other words, the validation function verifies if a given service execution reaches state C (success) or state F (failure) of our certification model in Definition 4. We note that the certification model of the service and the dependability property remain constant, while the index k may change over time. We also note that the service is static, while its context changes. In particular, k is an index referring to the service executions (i.e., the validation tests) used by the certification authority to verify the dependability property of the service under different contexts (e.g., using fault injection).

Example 5 Consider the certification model for operation `write` in Figure 6. The certification authority validates property p=(reliability, {mechanism=redundancy, no_of_server_instances=3, fault_type=server_crashes, recovery_time≤15ms}) by performing a sequence of tests driven by the service model of operation `write` in Figure 2(a). For each test iteration, the validation function f returns $true$ if all conditions in the execution path are satisfied and the service reaches state C; it returns $false$ and moves to state F from any of its states, otherwise.

The results of the validation function are used by the certification authority to estimate the values of R_iPr_{ij} in Definition 4, and L in Definition 5. To this aim, we introduce a *frequency log* that maintains f's results. A frequency log is a list of triplets $(k, \{v_k\}, s_i)$, where: k is the index of the test request in Definition 6; $\{v_k\}$ represents the attribute value(s) causing a transition to F; and s_i is the state of the certification model in which a transition to F is observed. We note that $\{v_k\}$ and s_i are empty if f returns $true$. We also note that state s_i is identified by observing the fault returned by the service under certification and by accessing the state of the service model in which the execution is currently blocked, since a guard that is linked to one or more policy conditions in the certification model is violated. Each probability R_iPr_{ij} can then be calculated, using the frequency log, as the number of successful transitions (s_i,s_j) over the total number of test requests reaching s_i. The total number of requests is such that each path in the model is tested for a given number of times. As an example, let us consider the certification model in Figure 6 and property p in Example 5, and suppose that the service fails to recover from a server crash in state s_{7C}; the frequency log registers (k,{no_of_server_instances=2,recovery_time>15ms},s_{7C}). On the basis of the values of R_iPr_{ij} estimated using the validation function, assurance level L in Definition 5 is quantified by performing the matrix operations described in Section 4, and used to characterize the dependability property of a service.

When M_{wsdl} is used, each most important operation (MIO) is validated individually, and assurance level is calculated for each MIO independently.

Let o_i be a MIO and L_{o_i} be its assurance level. A dependability certificate is issued to the service if the assurance level of each operation L_{o_i} is greater than or equal to a predefined threshold $\mathcal{T}\in[0,1]$. When M_{wscl} is used, the overall conversation is validated, and a certificate is issued to the service if assurance level L of the conversation is greater than or equal to $\mathcal{T}$. The dependability certificate is then of the form $\mathcal{C}(p, M, \{(o_i, L_{o_i})\})$, where: *i)* p represents the dependability property supported by the service; *ii)* $M\in\{M_{wsdl}, M_{wscl}\}$ is the service model; *iii)* $\{(o_i, L_{o_i})\}$ includes assurance level L_{o_i} with which each o_i supports p when a WSDL-based model is used; it contains a single pair $(-,L)$ when a WSCL-based model is used. We note that a service certified using M_{wscl} is first certified using M_{wsdl}.

5.2 Online Phase

The online phase starts immediately after the service provider deploys its certified service. In this phase, the certification authority continuously verifies the validity of the certificate issued to the service, since, in complex digital ecosystems, dependability properties may change over time, resulting in outdated certificates. For example, certified reliability and availability properties of the service may change if a replica failure or network congestion happens. To this aim, we introduce *Evaluation Body*, a component that is owned by the certification authority and placed in the system where the certified service is deployed, to monitor its dependability property. In particular, when the WSCL-based model is available, overall conversation is monitored using M_{wscl}; otherwise, each MIO is monitored individually using connected components $m_{wsdl}\in M_{wsdl}$. The results obtained by monitoring are then reflected on the corresponding certification model(s) M_{cert} to verify $Pol(p)$, and to update the assurance level(s) in the certificate at run-time.

Since the certification model generates all possible states of the service, the number of states can be extremely large. However, to monitor and verify dependability properties of a service, we do not need the complete Markov model. Therefore we derive a lightweight Markov model by reducing the original one while maintaining its accuracy. We note that this reduction is applied on the certification model to improve the performance of the monitoring process, and reduce requirements on the platform deploying the service. A reduced certification model $\tilde{M}_{cert}$ is formally defined as follows.

Definition 7 (Reduced certification model) Let M_{cert} be a certification model, a reduced certification model $\tilde{M}_{cert}$ is of the form $\tilde{M}_{cert}=\langle \mathcal{S}, s_1, C, F, \xrightarrow{c_{ij}}, R_i Pr_{ij}\rangle$, such that, $|\tilde{M}_{cert}(\mathcal{S})|<|M_{cert}(\mathcal{S})|$. For all validation tests, *i)* $f(ws, p, \tilde{M}_{cert}, k)=f(ws, p, M_{cert}, k)$ and *ii)* the frequency logs for $\tilde{M}_{cert}$ and M_{cert} are consistent.

The frequency logs are consistent if, for each entry $(k, \{v_k\}, s_i)$ generated using M_{cert}, there exists $(\tilde{k}, \{\tilde{v}_k\}, \tilde{s}_i)$ generated using $\tilde{M}_{cert}$, such that $k=\tilde{k}$,

$\{v_k\}=\{\tilde{v}_k\}$, and $s_i=\tilde{s}_i$ or $\tilde{s}_i$ is a combination of states including s_i. For example, states s_{7B} and s_{7C} of M_{cert} in Figure 6 can be combined to a single state $\tilde{s}_{7BC}$ to obtain a reduced Markov model $\tilde{M}_{cert}$. In $\tilde{M}_{cert}$, service moves from s_{7A} to $\tilde{s}_{7BC}$ following a combination of $[c_2]$ and $[c_3]$, and from $\tilde{s}_{7BC}$ to s_{7D} following a combination of $[c_4]$ and $[c_5]$. Transition $(\tilde{s}_{7BC}, F)$ is a combination of transitions (s_{7B}, F) and (s_{7C}, F) in M_{cert}. The results of f using $\tilde{M}_{cert}$ are then the same as the ones obtained using M_{cert}.

At run-time, the evaluation body monitors service executions by obtaining real-attribute values using the service model, and maps them to $\tilde{M}_{cert}$. A dependability certificate issued to a service remains valid if its real-attribute values satisfy the conditions in the policy with a given level of assurance. For each service execution verified using $f(ws, p, \tilde{M}_{cert}, k)$, the probability values in the original model M_{cert} (and matrix Q') must be updated using the results of f, and the assurance level of the service must be recomputed in order to verify if $L_{o_i}\geq\mathcal{T}$ for each service operation o_i in case of WSDL-based model and $L\geq\mathcal{T}$ in case of WSCL-based model. To this aim, as in the offline phase, we use a *frequency log*, storing f's results, within the evaluation body. We note that the source of failure or policy violation in M_{cert} can be precisely located using the service model, the frequency log, and $\tilde{M}_{cert}$. We also note that the update of matrix Q' is done periodically to preserve system performance.

We extend the notion of assurance level to support the validation process of the certification authority, and define a random variable L^t to characterize the dependability property of a service at run-time. For simplicity, in the remaining of this section, we use L^t to refer to the assurance level at time t for certification processes that rely on both WSDL-based and WSCL-based models. Given the time instant t at which the evaluation body starts the matrix update, L^t represents the assurance level of the service quantified by matrix Q', updated using the reduced Markov model $\tilde{M}_{cert}$ and the frequency log. Assurance level L^t observed by the evaluation body leads to the following conditions: *i)* $L^t\geq L^0$, where L^0 is the assurance level when the certificate was issued to the service in the offline phase. This implies that $L^t\geq\mathcal{T}$, that is, the assurance value of the service at run-time is still greater than the predefined threshold value $\mathcal{T}$, and the dependability certificate of the service remains valid; *ii)* $L^t<L^0$, in this case, the evaluation body first checks whether $L^t\geq\mathcal{T}$. If true, the certificate remains valid; otherwise, the certification authority either updates the dependability certificate or revokes it. We extend the definition of dependability certificate as $\mathcal{C}(p, M, \{(o_i, L^t_{o_i})\})$ to comply with dynamic changes in service dependability.

5.3 Dependability Certificate Life-cycle

The certificate life-cycle starts in the offline phase when the certification authority issues a certificate $\mathcal{C}$ to a service and marks it as *valid*. $\mathcal{C}$ is associated with a validity period t_e, where e is the expiration date of the certificate. During the online phase, the evaluation body monitors the service executions,

checks the certificate validity, and updates the assurance level in the certificate using real-attribute values. As long as $L^t \geq \mathcal{T}$ and $t < t_e$, for property p and model M, certificate $\mathcal{C}$ remains valid. The following situations can occur when $L^t < \mathcal{T}$ and $t < t_e$.

- The certification authority builds a new certification model for a new property $p_i \preceq_P p$, by relaxing some policy conditions. For example, if the original policy condition is *no_of_server_instances*$\geq$3, the new certification model may relax the condition as *no_of_server_instances*$\geq$2, and consider a new property with two replicas. Based on the new certification model, validation tests are performed on the service by monitoring real service executions; if $L^t \geq \mathcal{T}$, a *downgraded* certificate with property p_i is issued to the service.
- When a downgraded certificate cannot be generated, $\mathcal{C}$ is *revoked*.

At a given point in time, if the service with a downgraded certificate resumes (part of) correct functionality and satisfies dependability property p with $L^t \geq \mathcal{T}$, using the original certification model (e.g., the model integrating policy condition *no_of_server_instances*$\geq$3), an *upgraded* certificate is issued to the service. We note that, while the assurance level in the upgraded certificate can be higher than the one in the original certificate, the property can be at most the one in the original certificate. Finally, based on the validity time t_e, certificate $\mathcal{C}$ can be renewed as follows. The certification authority starts a renewal process at time $t_i < t_e$, where the exact time t_i depends on the considered scenario, to re-validate dependability property p, and in turn the certificate, for the service. If $L^{t_i} \geq \mathcal{T}$ holds, a renewed *valid* certificate is offered to the service, with a new initial assurance level L^0 and a new validity time t_e. Otherwise, if $L^{t_i} < \mathcal{T}$ or t_e expires, the certificate becomes *invalid*. We note that, at any point in time, a certificate can be either valid, invalid, upgraded, downgraded, or revoked. In the following, when clear from the context, we refer to valid, downgraded, and upgraded certificates as simply valid certificates.

6 Dependability Certificate-Based Service Selection

We aim to provide a solution where services can be searched and selected at run-time based on their dependability certificate and client's dependability requirements. To this aim, our service discovery component extends standard service registries *i)* to incorporate the dependability metadata in the form of certificates and *ii)* to support the matching and comparison processes described in the following of this section.

Let us consider a service registry that contains a set of services ws_j, each one having a dependability certificate $\mathcal{C}_j(p, M, \{(o_i, L^t_{o_i})\})$. A client can define its dependability requirements $Req(p, M, \{(o_i, L_{o_i})\})$ in terms of preferences on *i)* dependability property $Req.p$, *ii)* granularity of the service model used to validate and certify the service $Req.M \in \{M_{wsdl}, M_{wscl}\}$, and *iii)* assurance level $Req.\{(o_i, L_{o_i})\}$ for M_{wsdl} or $Req.(-, L)$ for M_{wscl}.

The matching process performs an automatic matching of client's requirements *Req* against valid dependability certificates $\mathcal{C}_j$ of services in the registry, and returns the set of services satisfying the specified requirements. The matching process implements a three-step process as follows [3].

– *Property match*: it selects services such that $Req.p \preceq_P \mathcal{C}_j.p$, using the hierarchy of dependability properties defined in Section 2.
– *Model match*: it selects services such that $Req.M \preceq_M \mathcal{C}_j.M$, that is, either $Req.M = \mathcal{C}_j.M$, or $Req.M = M_{wsdl}$ and $C_j.M = M_{wscl}$. The latter condition holds since a service certified for M_{wscl} is first certified for M_{wsdl}.
– *Assurance level match*: it selects services on the basis of the assurance level in the certificate. In case of WSDL-based model, a service is selected iff $Req.L_{o_i} \leq \mathcal{C}_j.L^t_{o_i}$ for each operation o_i. In case of WSCL-based model, a service is selected iff $Req.L \leq \mathcal{C}_j.L^t$.

The matching process returns a set *WS* of services compatible with client's preferences, according to property, model, and assurance level matches. The comparison process takes *WS* as input and transparently generates an ordering of services. The goal of this phase is to rank the shortlisted set of services in *WS* based on their certificates so as to facilitate the client in selecting the most appropriate service among the compatible ones. Given two services $ws_j, ws_k \in WS$ with certificates $\mathcal{C}_j$ and $\mathcal{C}_k$, respectively, the ordering of services is performed based on the hierarchical relationship among dependability properties, the model granularities, and the assurance level values. We note that, in some cases, there can be inconsistencies in the comparison (e.g., $\mathcal{C}_j.p \preceq_P \mathcal{C}_k.p$ and $\mathcal{C}_j.M \preceq_M \mathcal{C}_k.M$, but $\mathcal{C}_j.L^t \not\leq \mathcal{C}_k.L^t$). In this context, we assume a default precedence rule in which the property is more important than the model, and the model is more important than the assurance level. We note that different precedence rules can also be used based on client's preferences [3]. A (partially) ordered set $\overline{WS}$ of services in *WS* is returned to the client as the output of the comparison phase.

Example 6 Let us consider a client searching for a storage service at time t, and a service discovery with four storage services st_1, st_2, st_3, st_4, each one having a valid dependability certificate $\mathcal{C}_{st_1}$, $\mathcal{C}_{st_2}$, $\mathcal{C}_{st_3}$, $\mathcal{C}_{st_4}$. Table 2 presents the client's requirements *Req* and the four dependability certificates.

Upon receiving request *Req* from the client, the service discovery starts the three-step matching process. First, it matches the client's requirement on dependability property *Req.p* against the dependability property in the certificates. Here, service st_4 is filtered out because $Req.p \not\preceq_P \mathcal{C}_4.p$. The service discovery then performs a match on the model used to certify services. In this step, st_2 is not selected since $Req.M \not\preceq_M \mathcal{C}_2.M$. Finally, the matching process considers the assurance level and returns $WS = \{st_1, st_3\}$, since $Req.L \leq \mathcal{C}_1.L^t$ and $Req.L \leq \mathcal{C}_3.L^t$. The result of the matching process is the set of compatible services *WS* that is given as input to the comparison process. The comparison process then compares certificates C_{st_1} and C_{st_3}, and produces an ordered list $\overline{WS} = \{st_3, st_1\}$ since $C_{st_1}.p \preceq_P C_{st_3}.p$. Service st_3 is finally returned to the client as the most appropriate service that satisfies its preferences.

Table 2 Dependability certificates of the services in the registry, and client's requirements

DEPENDABILITY CERTIFICATES

	$\hat{p}$	A	M	L^t
$\mathcal{C}_{st_1}$	Reliability	mechanism=redundancy no_of_server_instances=3 fault_type=server_crashes	M_{wscl}	L^t=0.98
$\mathcal{C}_{st_2}$	Reliability	mechanism=redundancy no_of_server_instances=4 fault_type=server_crashes recovery_time≤15ms	M_{wsdl}	$L^t_{\texttt{read}}$ =0.96 $L^t_{\texttt{write}}$=0.95
$\mathcal{C}_{st_3}$	Reliability	mechanism=redundancy no_of_server_instances=4 fault_type=server_crashes recovery_time≤15ms	M_{wscl}	L^t=0.90
$\mathcal{C}_{st_4}$	Availability	mechanism=redundancy recovery_time≤15ms	M_{wscl}	L^t=0.92

CLIENT'S REQUIREMENTS

	$\hat{p}$	A	M	L
Req	Reliability	mechanism=redundancy no_of_server_instances≥3 fault_type=server_crashes	M_{wscl}	L≥0.85

We extend matching and comparison processes to complement our certification scheme and, provide a two-phase service selection solution. In the first phase, a static service selection is performed when the client sends a request to the service discovery. The second phase starts when the client selects a service $ws_j \in \overline{WS}$. In this phase, service discovery performs constant monitoring of the certificate status for ws_j. If a certificate is downgraded, revoked, or moves to invalid state, the service discovery triggers the matching and comparison processes and replaces the originally selected service with a new, compatible, service $ws_k \in \overline{WS}$. The second phase is transparent to the client and allows our solution to ensure clients requirements also during run-time.

7 Certifying Business Processes

A service provider can implement its business process as a composition of different services, provided by different suppliers. To this aim, it defines a *template* specifying the order in which service operations must be called, the data to be exchanged in each phase of the composite service workflow, and the conditions under which a given service instance must be integrated within the business process. In this paper we consider Business Process Execution Language (BPEL), a de-facto standard for web service composition [1]. BPEL templates define executable processes using XML and mainly consider functionality requirements in the selection of component services to be integrated in a business process. Our goal is to extend the modeling approach and the certification scheme in this paper to give a first solution to the run-time certification of dependability properties for composite services.

7.1 Modeling a Service Composition

Given the BPEL template and the WSDL-based model of partner services, we define the BPEL-based model M_{bpel} of the composition, which is then used to certify the dependability property of the business process. To this aim, we extend the set $\mathcal{O}$ of operators in Section 3.2 with the parallel operator $\oplus$, that is, $\mathcal{O}$=$\{\odot,\otimes,\oplus\}$. The parallel operator is used to model processes involving the simultaneous invocation of different operations; for instance, in eShop, operation `browseItems` of three vendor services are invoked in parallel. Operators in $\mathcal{O}$ are recursively applied on the connected components m_{wsdl} of partner services to incrementally derive M_{bpel} as follows.

$$M_{bpel} = m_{wsdl} \mid M_{bpel} \odot M_{bpel} \mid M_{bpel} \otimes M_{bpel} \mid M_{bpel} \oplus M_{bpel}$$

The BPEL-based model can be formally defined as follows.

Definition 8 (BPEL-based model) Let $\mathcal{M}$ be the set of service models, BPEL-based model $M_{bpel}\in\mathcal{M}$ of a service is an STS $\langle\mathcal{S}, s_1, \mathcal{V}, \mathcal{I}, \mathcal{A}, \rightarrow\rangle$ (see Definition 1), where $\mathcal{S}$ is the union of all the states of the WSDL-based models of partner services integrated using the operators in $\mathcal{O}$=$\{\odot,\otimes,\oplus\}$, and $\mathcal{A}$ represents the set of service operations selected and integrated in the business process.

We note that given two BPEL-based models M_1 and M_2 composed using the parallel operator $\oplus$, the initial states of the two models are represented as a single state where the input is distributed to the two parallel flows; similarly, the final states of the two models are represented as a single state where the results of the two parallel executions are combined. For sequence $\odot$ and alternative $\otimes$ operators, the STS-based model is built as discussed in Section 3.2 for M_{wscl}. The conceptual difference between the WSCL-based and BPEL-based models is that the former considers operations of a single service, while the latter of different services. As for the WSCL-based model, the set $\mathcal{S}$ of states in M_{bpel} can also be extended when the source code of the service operations is available, and there is an implicit transformation from M_{wsdl} to M_{bpel} on the basis of the WSDL-based model of partner services and $\mathcal{O}$.

7.2 Certification Scheme for Business Processes

The dimension of failures in business processes is significantly different from monolithic services, since business process dependability is affected by the composition protocol and partner services. This implies that business process owner (the client in our framework) must utilize those partner services that not only satisfy its functional requirements, but also its requirements on dependability properties. We therefore require the client to: *i)* define the business process in the form of a BPEL template, *ii)* select dependability property p to be certified for its business process, *iii)* extend the BPEL template with a set of requirements Req_j on the dependability of each partner service ws_j to

be integrated. In the following, for the sake of clarity, we assume Req_j to only include requirements on property $Req_j.p$.

When requirements $Req_j.p$ are defined in the BPEL template, the client can use the certificate-based matching and comparison processes in Section 6 to select appropriate partner services. The chosen services can then be integrated and orchestrated to realize the business process (which we call BPEL *instance*). We note that the BPEL instance produced in this manner will hold property p, or a stronger property, since our matching and comparison processes select services ws_j considering $Req_j.p$ as the lower bound. To this aim, we assume a common ontology specifying rules for property composition. This ontology can drive the client in the specification of BPEL templates annotated with suitable requirements for the certification of specific properties of composite services.

The certification process starts when a client releases its BPEL instance, and requests the certification authority to certify property p for the business process. Differently from the certification of single services, the certification authority cannot determine assurance level and certify a composite service a priori during the offline phase, because the integration of component services is performed at run-time. However, to avoid downtimes in service certification, we first estimate assurance level L_{bpel} of a composition at run-time, according to the following rules.

- When two operations o_i and o_j are composed in a sequence, the assurance level of the sequence is L_{ij}=$L^t_{o_i}$$*$$L^t_{o_j}$, and o_i and o_j are considered as a single operation o_{ij}. This rule is based on the assumption that partner services perform their operations independently of each other. For example, suppose operation `shipItems` of shipping service sh and operation `write` of storage service st are invoked in a sequence, the assurance level of their composition is $L^t_{sh}$$*$$L^t_{st}$.
- When two operations o_i and o_j are composed in a parallel or alternative, the assurance level is L_{ij}=min($L^t_{o_i}, L^t_{o_j}$), and o_i and o_j are considered as a single operation o_{ij}. For example, when eShop invokes operation `browseItems` from three independent vendors v_1, v_2 and v_3 in parallel, it can perform its correct functionality (e.g., it generates a table of items returned from all three vendors) only if all three vendors behave correctly. If either vendors fail, the dependability of eShop is affected. Therefore, the assurance level of the composition is min($L^t_{v_1}, L^t_{v_2}, L^t_{v_3}$).

The above rules are recursively applied by the certification authority to a BPEL instance as follows: *i)* all pairs in a sequence are considered; *ii)* when no sequences are left, an alternative or parallel is considered; *iii)* points *i)* and *ii)* are repeated until a single operation o is left. The certification authority then estimates assurance level L_{bpel} of the composition and awards a *temporary* dependability certificate $\mathcal{C}(p, M_{bpel}, L_{bpel})$, if $L_{bpel} \geq \mathcal{T}$. We note that, since dependability requirements $Req_j.p$ given as input to the matching and comparison processes represent lower bounds for service selection, the certification authority could include a stronger property p_j (i.e., $Req_j.p \preceq_P p_j$) in the certifi-

cate. As an example, consider a client specifying its requirements for two partner services in a sequence as $Req_j.p$=(reliability, {mechanism=redundancy, no_of_server_instances=2, fault_type=server_crashes}); then, suppose that the matching and comparison processes return two services with property p_j=(reliability,{mechanism=redundancy, no_of_server_instances=4, fault_type=server_crashes}), with $Req_j.p \preceq_P p_j$. Clearly, if $L_{bpel} \geq \mathcal{T}$, the composition is certified for property p_j.

After releasing the temporary certificate, the certification authority first produces the Markov-based certification model as discussed in Section 4.1. It then monitors the business process by observing executions of the BPEL instance, maintains the results in the frequency log, and updates the Q' matrix as discussed in Section 5.2. When a given (set of) quality metric has been satisfied by service executions (e.g., all the execution paths in the BPEL instance have been invoked a sufficient number of times or a given coverage of the service model has been achieved), the certification authority calculates the new assurance level L^t_{bpel} at time t using the approach in Section 5.2. If $L^t_{bpel} \geq \mathcal{T}$, the temporary certificate becomes a valid certificate with assurance level L^t_{bpel}, otherwise it is revoked. It is important to note that in case a service ws_j in a composition becomes unavailable or its certificate violates $Req_j.p$, the selection process in Section 6 is executed to substitute ws_j with another candidate ws_k on the basis of the dependability requirement $Req_j.p$ in the BPEL template. As soon as ws_k has been selected and integrated, the process restarts with the generation of a temporary certificate.

Example 7 Let us consider our reference scenario in which eShop composes three vendor services (v_1, v_2, v_3), two shipment services (sh_1, sh_2), and a single storage service (st). eShop defines a BPEL template and requires property $Req.p$=(reliability, {mechanism=redundancy, no_of_server_instances≥3, fault_type=server_crashes}) for all services to be composed. It then uses the matching and comparison processes in Section 6 to select them. Figure 7 illustrates the BPEL instance addressing the above requirements. Here, for simplicity, we assume that each selected service (v_1, v_2, v_3, sh_1, sh_2, st) has been certified for $Req.p$, with M_{wsdl}, and with the same assurance level L^t for all its operations. We consider the following values for L^t: $L^t_{v_1}$=0.95, $L^t_{v_2}$=0.98, $L^t_{v_3}$=0.92, $L^t_{sh_1}$=1, $L^t_{sh_2}$=0.95, and L^t_{st}=0.96.

The certification of the BPEL instance starts with the generation of a temporary certificate. Since there are no sequences, the certification authority first considers operation `browseItems` from vendor services that are executed in parallel, and estimates the assurance level as $L_{v_{123}}=\min(0.95, 0.98, 0.92)=0.92$. The same process applies to operation `buyGoods` of vendor services composed in an alternative, where $L_{v'_{123}}=\min(0.95, 0.98, 0.92)=0.92$. The operations `browseItems` and `buyGoods` are further composed in a sequence, and the overall assurance level is $L_{v_{123},v'_{123}}=0.92 * 0.92=0.8464$. The shipment services are then invoked in an alternative, and the assurance level estimated as $L_{sh_{12}}=\min(1, 0.95)=0.95$. Finally, eShop composes the vendor, shipment, and storage services in a sequence. The assurance level of eShop is estimated as

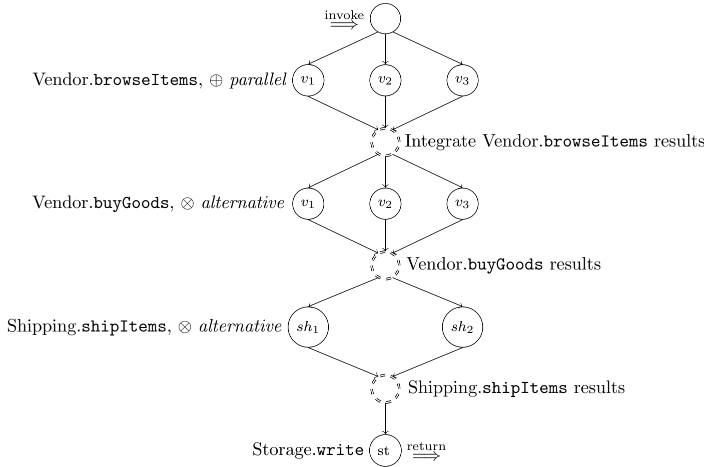


Fig. 7 An example of a composition in the eShop business process

L_{bpel}=($L_{v_{123},v'_{123}}$*$L_{sh_{12}}$)*L_{st}=(0.8464*0.95)*0.96=0.772, where the assurance level of the sequence between vendor and shipment services is first estimated, and the overall assurance level calculated as the sequence between the sequence of vendor and shipment services and the storage service. The certification authority generates a temporary certificate and issues it to eShop, if $L_{bpel} \geq \mathcal{T}$.

After the release of a temporary certificate, eShop is continuously monitored by the certification authority and a valid certificate $\mathcal{C}(p, M_{bpel}, L^t_{bpel})$ at time t is awarded to it, if it satisfies property p with assurance level $L^t_{bpel} \geq \mathcal{T}$. When one or more among v_1, v_2, v_3, sh_1, sh_2, st become unavailable or their certificates violate client's requirements, the certificate for the composition is revoked and the validation process resumes.

8 Related Work

The proposed certification solution has multi-disciplinary roots involving areas of SOA, system modeling, testing, and certification. In this section, we discuss some related work corresponding to these areas.

An important line of research concerns definition of modeling approaches for automatic generation of test cases and, validation of functional and QoS parameters for services and service compositions. Salva and Rabhi [30] propose an approach to test the robustness of a service by automatically generating test cases from its WSDL interface. Frantzen et al. [14] define an approach to model service coordination using an STS and automatically generate run-time test cases. Salva et al. [29] propose a testing method based on STSs and security rules for stateful web services. Ding et al. [13] model failure behavior using nonhomogeneous poisson process and compute the overall system reliability through the reliability of partner link, port type, and operation. Riccobene

et al. [28] use architecture- and path-based reliability models to predict the reliability of an SCA-ASM component model, and of the SCA assembly modeling a service orchestration, by considering failures specific to the nature of ASMs. Bentakouk et al. [8] propose a testing solution that uses an SMT solver to check the conformance of a composite service against its specifications. In contrast to the above works, we use formal modeling to validate and certify dependability properties of services. Mateescu and Rampacek [23] define an approach for modeling business processes and web services described in BPEL using process algebraic rules. Betakauk et al. [7] propose a framework for testing service orchestrations. The orchestration specification is translated into an STS, and then, a Symbolic Execution Tree (SET) is computed. SET supports retrieval of STS execution semantics and, for a given coverage criterion, generation of a set of execution paths which are run by a test oracle against the orchestration implementation. Pathak et al. [27] define a framework that models service compositions as STSs starting from UML state machines. Similarly to the above approaches, we model services as state automata (using STSs) and apply a derivative approach to generate the models for service compositions. However, our solution generates dependability certificates for service compositions using certificates of individual services.

Definition of service certification schemes is the most relevant aspect to our work. Kourtesis et al. [22] present a solution using Stream X-machines to improve the reliability of SOA environments; they evaluate if the service is functionally equivalent to its specifications, and award a certificate to it. Anisetti et al. [2, 4] proposes a model-driven test-based security certification scheme for (composite) services. This solution allows clients to select services on the basis of their security preferences [3]. The work in [9] presents a test-based reliability certification scheme for services that provides an a priori validation of services based on reliability patterns, and a posteriori testing using a set of metrics. Our work, instead, quantitatively evaluates dependability properties of (composite) services and supports run-time validation of certificates.

The approach of probabilistically estimating the reliability of a software using Markov chains has been studied in the past. Mustafiz et al. [25] propose an approach to identify reliable ways of designing the interactions in a system and assigning probability values to each interaction measuring its success. Cheung [10] claims that reliability of a software depends on the reliability of its components and the probabilistic distribution of their utilization. A Markov model is then used to measure the reliability and effects of failures on the system with respect to a user environment. Other Markovian model-based approaches to evaluate system reliability have been proposed (e.g., [16, 24]).

9 Conclusions

We presented a dependability certification scheme in which a machine-readable certificate is issued to the service after validating its dependability properties using Markov chains. The service is then continuously monitored at run-time

to verify the validity of the issued certificate. The proposed solution can be integrated within existing service-oriented architectures: it allows clients to search and select services with a given set of dependability properties and ensures that client's requirements are addressed at run-time. Finally, building on our certificate-driven selection solution for monolithic services, we presented a modeling and certification solution for business processes.

Acknowledgments

This work was partly funded by the European Commission under the project ASSERT4SOA (contract n. FP7-257351), the Italian Ministry of Research within PRIN project "GenData 2020" (2010RTFWBH), and by Google, under the Google Research Award program.

References

1. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. OASIS (2007). http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
2. Anisetti, M., Ardagna, C., Damiani, E.: Security certification of composite services: A test-based approach. In: Proc. of 20th IEEE Int'l Conference on Web Services (2013)
3. Anisetti, M., Ardagna, C., Damiani, E., Maggesi, J.: Security certification-aware service discovery and selection. In: Proc. of 5th Int'l Conference on Service-Oriented Computing and Applications (2012)
4. Anisetti, M., Ardagna, C., Damiani, E., Saonara, F.: A test-based security certification scheme for web services. ACM Transactions on the Web **7**(2), 5 (2013)
5. Ardagna, C., Damiani, E., Jhawar, R., Piuri, V.: A model-based approach to reliability certification of services. In: Proc. of 6th Int'l Conference on Digital Ecosystem Technologies - Complex Environment Engineering (2012)
6. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing **1**(1), 11–33 (2004)
7. Bentakouk, L., Poizat, P., Zaïdi, F.: A formal framework for service orchestration testing based on symbolic transition systems. In: Proc. of the 21st IFIP WG 6.1 Int'l Conference on Testing of Software and Communication Systems (2009)
8. Bentakouk, L., Poizat, P., Zaïdi, F.: Checking the behavioral conformance of web services with symbolic testing and an SMT solver. In: Proc. of 5th Int'l Conference on Tests and Proofs (2011)
9. Buckley, I., et al.: Towards pattern-based reliability certification of services. In: Proc. of 1st Int'l Symposium on Secure Virtual Infrastructures (2011)
10. Cheung, R.C.: A user-oriented software reliability model. IEEE Transactions on Software Engineering **6**, 118–125 (1980)
11. Damiani, E., Ardagna, C., El Ioini, N. (eds.): Open source systems security certification. Springer, New York, NY, USA (2009)
12. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P.: Securing SOAP e-services. Int'l Journal of Information Security **1**(2), 100–115 (2002)
13. Ding, Z., Jiang, M., Kandel, A.: Port-based reliability computing for service composition. IEEE Transactions on Services Computing **5**(3), 422–436 (2012)
14. Frantzen, L., Tretmans, J., de Vries, R.: Towards model-based testing of web services. In: Proc. of the Int'l Workshop on Web Services - Modeling and Testing (2006)
15. Herrmann, D.: Using the common criteria for IT security evaluation. Auerbach Publications (2002)

16. Iyer, S., Nakayama, M., Gerbessiotis, A.: A markovian dependability model with cascading failures. IEEE Transactions on Computers **58**, 1238–1249 (2009)
17. Jhawar, R., Piuri, V.: Adaptive resource management for balancing availability and performance in cloud computing. In: Proc. of 10th Int'l Conference on Security and Cryptography (2013)
18. Jhawar, R., Piuri, V.: Fault tolerance and resilience in cloud computing environments. In: Computer and Information Security Handbook, 2nd Edition. Morgan Kaufmann (2013)
19. Jhawar, R., Piuri, V., Samarati, P.: Supporting security requirements for resource management in cloud computing. In: Proc. of 15th IEEE Int'l Conference on Computational Science and Engineering (2012)
20. Jhawar, R., Piuri, V., Santambrogio, M.: Fault tolerance management in cloud computing: A system-level perspective. IEEE Systems Journal **7**(2), 288–297 (2013)
21. Keum, C., Kang, S., Ko, I.Y., Baik, J., Choi, Y.I.: Generating test cases for web services using extended finite state machine. In: Proc. of 18th IFIP Int'l Conference on Testing Communicating Systems (2006)
22. Kourtesis, D., Ramollari, E., Dranidis, D., Paraskakis, I.: Increased reliability in SOA environments through registry-based conformance testing of web services. Production Planning & Control **21**(2), 130–144 (2010)
23. Mateescu, R., Rampacek, S.: Formal modeling and discrete-time analysis of BPEL web services. In: Advances in Enterprise Engineering I, *Lecture Notes in Business Information Processing*, vol. 10, pp. 179–193. Springer Berlin Heidelberg (2008)
24. Muppala, J., Malhotra, M., Trivedi, K.: Markov dependability models of complex systems: Analysis techniques. Reliability and Maintenance of Complex Systems, NATO ASI Series F: Computer and Systems Sciences **154**, 442–486 (1996)
25. Mustafiz, S., Sun, X., Kienzle, J., Vangheluwe, H.: Model-driven assessment of system dependability. Software and System Modeling **7**(4), 487–502 (2008)
26. Papazoglou, M.: Web services and business transactions. World Wide Web **6**, 49–91 (2003)
27. Pathak, J., Basu, S., Honavar, V.: Modeling web service composition using symbolic transition systems. In: Proc. of AAAI Workshop on AI-Driven Technologies for Service-Oriented Computing (2006)
28. Riccobene, E., Potena, P., Scandurra, P.: Reliability prediction for service component architectures with the SCA-ASM component model. In: Proc. of 38th EUROMICRO Conference on Software Engineering and Advanced Applications (2012)
29. Salva, S., Laurencot, P., Rabhi, I.: An approach dedicated for web service security testing. In: Proc. of 5th Int'l Conference on Software Engineering Advances (2010)
30. Salva, S., Rabhi, I.: Automatic web service robustness testing from WSDL descriptions. In: Proc. of 12th European Workshop on Dependable Computing (2009)
31. Samarati, P., De Capitani di Vimercati, S.: Data protection in outsourcing scenarios: Issues and directions. In: Proc. of 5th ACM Symposium on Information, Computer and Communications Security. Beijing, China (2010)
32. Tretmans, J.: Model-based testing and some steps towards test-based modelling. In: Proc. of 11th Int'l School on Formal Methods for Eternal Networked Software Systems (2011)
33. Trivedi, K., et al.: Dependability and security models. In: Proc. of 7th Int'l Workshop on Design of Reliable Communication Networks (2009)
34. USA Department of Defence: Department Of Defense Trusted Computer System Evaluation Criteria (1985). http://csrc.nist.gov/publications/secpubs/rainbow/std001.txt
35. van Veenendaal, E.: Standard glossary of terms used in Software Testing Version 2.2. International Software Testing Qualifications Board (2012). http://www.astqb.org/documents/ISTQB_glossary_of_testing_terms_2.2.pdf, Accessed in August 2013